VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Software pro tvorbu rejstříku v LaTeXu

LaTeX Indexing Software

2016 Tomáš Rapi

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Rapi**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Software pro tvorbu rejstříku v LaTeXu
LaTeX Indexing Software

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit moderní obdobu programu MakeIndex, který je používám při zpracování dokumentů v LaTeXu pro tvorbu rejstříků. Současné implementace mají zásadní problémy při zpracování češtiny, obzvláště v kódování UTF8. Nová implementace by měla proběhnout podle zásad OOP v prostředí .NET frameworku.

Požadavky na vypracování:
1. Proveďte rešerši k tvorbě rejstříku v LaTeXu (MakeIndex, Xindy atd.).
2. Navrhněte funckionalitu nového software pro tvorbu rejstříku.
3. Zpracujte objektově orientovaný návrh software.
4. Takto navržený software implementujte v prostředí .NET a v jazyce C#.
5. Zpracujte programátorskou a uživatelskou dokumentaci (ukázky použití).

Seznam doporučené odborné literatury:

[1] Kopka H., Daly P.W.: LaTeX kompletní průvodce. Computer Press, 2004

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 01.09.2014
Datum odevzdání: 15.07.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 10. července 2016 .....

Abstrakt

Tato bakalářská práce shrnuje současnou situaci v oblasti softwaru k tvorbě rejstříků pro typografický systém LaTeX. Dnes neexistuje program, který by dokázal pracovat s *UTF-8* a byl spustitelný pod operačním systémem *Windows*. Proto je zde vypracovaný nový program pro generování rejstříků s názvem *SharpIndex*. *SharpIndex* je napsaný v jazyce *C#* s použitím technologií *.NET*. Pro správné řazení podle české abecedy program splňuje normu *ČSN 97 6030* [2], která stanovuje pravidla abecedního zpracování počítačového textu. Uživatelské rozhraní programu tvoří příkazová řádka, pro detailnější nastavení využívá externího *XML* souboru. Pro ukázku a porozumění funkčnosti jsou vypracovány i testy, ukázkové použití a dokumentace.

Klíčová slova: tvorba rejstříků, LaTeX, ČSN 97 6030

Abstract

This bachelor thesis summarises the current situation in the field of indexing software for the typographic system LaTeX. There is no program today that can work with *UTF-8* and can be run on the operating system *Windows*. This is why there is a newly created program for generating indexes called *SharpIndex*. *SharpIndex* is written in *C#* programming language and uses the *.NET* technology. This program meets the *ČSN 97 6030* [2] standard, which determines the rules of alphabetical ordering in computer texts. The user interface of the program is the command line and for more detailed settings it uses an external *XML* file. The thesis includes tests, samples and documentation for a better understanding of the program's functionality.

Keywords: creating index, LaTeX, ČSN 97 6030

Obsah

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době existuje mnoho různých textových editorů typu *WYSIWYG*. Tyto editory jsou vhodné převážně pro kratší texty s menší mírou provázanosti, menšími nároky na formátování a přenositelnost. Při tvorbě rozsáhlých textů, je vhodné použít sofistikovanější nástroj, který není typu *WYSIWYG*. Velmi rozšířený a populární nástroj pro tvorbu knih či matematických publikací, je typograficky systém TEX, případně LATEX. TEX vyvinul v 70. letech 20. století *Donald E. Knuth*, profesor na univerzitě ve Stanfordu, USA. Jeho cílem bylo zvýšit kvalitu psaných skript a publikací na univerzitě. Čistý TEX, tzv. *Plain* TEX, obsahuje zhruba 300 různých příkazů a stal se základním formátem TEXu. Bohužel *Plain* TEX není srozumitelný pro příležitostného uživatele. Z tohoto důvodu roku 1985 vytvořil *Leslie Lamport* nadstavbu pro TEX, kterou pojmenoval LATEX. Jedná se o množinu maker, které vytvářejí uživatelské rozhraní pro psaní v TEXu. Díky tomuto kroku je pro nezkušené uživatele rychlejší a jednodušší naučit se nástroj používat. Veškeré makra přitom vycházejí ze základního formátu *Plain* TEX. Každé LATEX makro je tedy i TEX makro. Z tohoto důvodu je tedy snadné tento programovací jazyk rozšiřovat a vyvíjet. Oficiální skupina, která se stará o rozvíjení LATEXu, je LATEX3. Tato skupina vedená *Frankem Mittelbachem*, má za cíl sjednotit všechny verze LATEXu. Současná verze nese označení LATEX2e. Nová verze systému LATEX3 je stále ve vývoji.

Mimo oficiálních tvůrců samozřejmě existují i menší skupiny, které se starají o určitou část typografického systému, kterou se snaží přizpůsobit či vylepšit. Vznikají tak nová makra, balíčky pro úpravu vzhledu, editory, programy pro tvorbu rejstříku či programy pro podporu regionálně a národnostně používaných zvyklostí. Tématem této bakalářské práce je právě tvorba rejstříků pro LATEX.

2 Jak LaTeX pracuje s rejstříky

LaTeX negeneruje rejstříky zcela automaticky. Je na autorovi, aby v textu pomocí příkazu `\index` označil výrazy, které se mají indexovat. Tento příkaz přijímá jako parametr slovo, jež se má indexovat. LaTeX dovoluje indexovat jak slova složená z písmen, číslic, speciálních znaků, tak i příkazy jako je například `\LaTeX`. Příkaz `\index` může obsahovat tři speciální znaky „!“, „@“ a „|“. Znak „!“ se používá pro oddělení podkategorií. Při indexování je dovoleno vytvářet maximálně 2 podkategorie. Příkaz se pak zapíše jako `\index\{USA!New York!město}` nebo `\index{USA!New York!stát}`. Příklad takovéhoto použití podkategorií je v tabulce 1, kde je zobrazena část souboru `.idx` a k němu zpracovaný soubor `.ind`. Znak „@“ umožňuje definovat, podle kterého hesla se má třídit a které heslo má být vypsáno do rejstříku. Index lze tedy zapsat jako `\index{latex@\LaTeX}`. Heslo bude seřazeno podle *latex*, ale do rejstříku bude vypsáno *LaTeX*. Znak „|“ se používá pro vymezení intervalu. V případě, že se na stránce 20 vyskytne levý indikátor typu hyperpage `\index{oblečení|(hyperpage}` a na stránce 26 pravý `\index{oblečení|)hyperpage}`, bude to znamenat, že položka rejstříku *oblečení* se vyskytuje na stranách 20 až 26. Znaky „!“ , „|“ a „@“, jež nemají mít speciální funkci, je nutné zapsat s uvozovkami před daným znakem (vykřičník se píše takto "!).

Proto ať LaTeX vytváří rejstřík, je dále nutné přidat v preambuli dokumentu (před `\documentclass{třída}`) příkaz `\makeindex`. Takto označený dokument při zpracování LaTeXem vygeneruje soubor, jež bude mít shodné jméno s hlavním dokumentem, jen s tím rozdílem, že bude mít příponu `.idx`. Uvnitř tohoto souboru budou na jednotlivých řádcích vypsány položky rejstříku spolu s číslem stránky na kterém se vyskytují, ve formátu `\indexentry {USA!New York!město} {68}`. Takto vysázený seznam není setříděný a slouží pouze jako vstupní soubor pro programy, které generují rejstříky. Jak bylo tedy již zmíněno, LaTeX nevytváří a negeneruje rejstříky sám. K tomuto účelu slouží externí programy, které budou podrobněji popsány v kapitolách 3.1, 3.2, 3.3 a 3.4. Tyto nástroje vytvoří soubor s příponou `.ind`, kde jsou jednotlivá hesla setříděná. Uvnitř tohoto souboru se vyskytují příkazy `\begin{theindex}`, `\end{theindex}`, `\item`, `\subitem` a `\subsubitem`. Jednotlivé položky `\item` jsou setříděny. Pokud `\item` obsahuje `\subitem`, jednotlivé `\subitem` jsou lokálně setříděny. Stejný princip se aplikuje i na `\subsubitem`. V souboru se navíc může vyskytovat `\indexspace`, který vloží mezeru pro logické oddělení různých částí rejstříku.

LaTeX při sestavování dokumentu kontroluje, zda existuje `.ind` soubor a pokud ano, použije jej pro vysázení rejstříku. V tabulce 2 je vlevo ukázka `.ind` souboru a vpravo jak by takovýto soubor LaTeX vysázel jako rejstřík. Podrobnější informace o možnostech tvorby rejstříků naleznete v [1].

```
\indexentry {USA!New York!stát}{35}
\indexentry {USA!New York!město}{55}
\indexentry {USA!Las Vegas}{119}
\indexentry {Kanada!Ottawa}{128}
\indexentry {Kanada!Ottawa}{131}
\indexentry {USA!Las Vegas}{135}
\indexentry {USA!Las Vegas}{137}
```

```
\begin{theindex}
\item Kanada
\subitem Ottawa, 128, 131
\indexspace
\item USA
\subitem Las Vegas, 119, 135, 167
\subitem New York
\subsubitem město, 55
\subsubitem stát, 35
\end{theindex}
```

Tabulka 1: Ukázka .idx (vlevo) a příslušného .ind souboru (vpravo).

```
\begin{theindex}
\item Kanada
\subitem Ottawa, 128, 131
\indexspace
\item USA
\subitem Las Vegas, 119, 135, 167
\subitem New York
\subsubitem město, 55
\subsubitem stát, 35
\end{theindex}
```

Kanada
    Ottawa, 128, 131

USA
    Las Vegas, 119, 135, 167
    New York
        Město, 55
        Stát, 35

Tabulka 2: Ukázka vysázení rejstříku (vpravo) z .ind souboru (vlevo).

3 Programy které generují rejstříky

Od doby vzniku systému LaTeX uplynulo více než 30 let a za tuto dobu se vystřídalo velké množství technologií, operačních systémů, programovacích jazyků či znakových sad. I přes to, že skupina LaTeX3 se snaží LaTeX neustále zdokonalovat, některé části zůstávají neaktualizované. Do této oblasti patří generátory rejstříků, obzvláště pak ty s národnostně specifickými požadavky. Základním úkolem těchto programů je vygenerovat `.ind` soubor, viz. 3, který LaTeX umí vysázet do dokumentu v místě, kde se nachází příkaz `\printindex`.

3.1 MakeIndex

MakeIndex vytvořil Chen & Harrison international Systems, Inc, v roce 1989. Jedná se o program napsaný v jazyce C, který vytvoří rejstřík ze vstupního `.idx` souboru, jež je generován LaTeXem. Velkou nevýhodou je podpora pouze charsetu *Basic Latin*, prvního bloku *Unicode* standartu, spolu se speciálními znaky obsaženými v německém jazyce. *MakeIndex* se obsluhuje a konfiguruje převážně pomocí konzole. Tento program nebyl navržený tak, aby jej bylo snadné rozšířit o nové funkcionality. To mělo za následek, že jádro programu zůstalo stejné a vzniklo jen pár upravených verzí programu, které nepřinesly žádné větší vylepšení nebo rozšíření. *MakeIndex* třídil znaky podle pořadí v *ASCII* tabulce, tedy nejdříve mezeru a speciální znaky, pak číslice, velká písmena a nakonec malá písmena. Existují pouze dva způsoby, jak měnit vstupy a výstupy tohoto programu. Parametry z příkazové řádky a pomocí *index style* souboru s příponou `.sty`. Z příkazové řádky šlo zapnout ořezávání mezer u slov, ignorovat znak mezera při abecedním řazení, použít pravidla pro německé řazení slov nebo načtení již zmíněného *index style* souboru. Hlavním smyslem souboru *index style* bylo udržení kompatibility mezi systémy TeX a LaTeX spolu s možností nastavit nové či změnit stávající klíčové znaky „!“, „@“ a „|“. Například TeX používá klíčové slovo `\glossary` a LaTeX `\indexentry`. *MakeIndex* má velmi omezené možnosti z hlediska parametrizace, nicméně uživatelé, kteří používali pouze *Basic Latin* charset, neměli moc důvodů, proč používat jiný program. *MakeIndex* je velmi rychlý a jednoduchý program.

3.2 Xindy

S nástupem nových druhů kódování, potřeby třídit hesla rejstříku podle regionálně specifických norem, byla v roce 1995 vytvořena první verze programu *Xindy*. *Xindy* byl vytvořený *Rogerem Kehrem*, jako studentský projekt na Darmstadt University of Technology pod vedením Joachima Schroda. *Xindy* přináší mnoho nových funkcí, které v *MakeIndexu* chyběly. Program byl rozdělen do jednotlivých modulů, aby byla jednodušší jeho rozšiřitelnost a konfigurace. Modul *base* převážně slouží k nastavení ostatních použitých modulů. Modul *class*, ve kterém se nastavuje použité číslování stránek (arabské číslice, římské nebo písmena řecké abecedy). Modul *lang*, ve kterém jsou popsány abecedy jazyků, včetně jejich třídící posloupnosti pro různá kódování, *Windows-1250*, *UTF-8* a jiné. Modul *ord*, ve kterém je možné nastavit pravidla pro třídění, která jsou nadřazená pra-

```
\begin{theindex}

\item automatic understanding, 119 – 135, 167

\indexspace

\item block
\subitem importance, 47
\subitem symbol, 20
\item boolean factor analysis, 85

\end{theindex}
```

Tabulka 3: Soubor .ind vytvořený indexovacím programem.

vidlům v modulu *lang*. V modulu *rules* jsou pravidla pro transliteraci a převod malých a velkých písmen pro různá použitá kódování. Modul *styles* slouží k definování vstupu a výstupu, možnosti ignorování specifických znaků, formátování *lettergroups* a mnoho dalších detailů. Pomocí tohoto modulu, dle slov tvůrců, lze *Xindy* překonfigurovat tak, ať zpracovává i jiné soubory než `.idx`, například *XML* či *HTML*. Poslední modul *tex*, slouží pro převod maker zpět na osmibitové znaky, například `\texttrademark` na ™.

Xindy je robustnější a propracovanější než *MakeIndex* téměř ve všech směrech. Jediným velkým nedostatkem *Xindy* je fakt, že funguje jen pod OS *Linux*. Původním záměrem bylo rozšířit distribuci na všechny běžně používané operační systémy, nicméně funkční verze byla vytvořena pouze pro *Linux*. Verze pro *Windows* či *MacOS* neexistuje. Vývoj tohoto programu byl ukončen, nové oficiální verze se neplánují.

3.3 CsIndex

Tento program napsal *Zdeněk Wágner* roku 1992 a v současnosti je spravován skupinou *www.ctan.org*. *CsIndex* obsahuje jádro programu *MakeIndex v2.11* a podporuje české abecední řazení podle normy ČSN 01 0181 z roku 1978. Tato norma již není platná a byla nahrazena normou ČSN 97 6030 z roku 1994 [2]. Program byl doplněn o nové funkce pro česky a slovensky psané texty, které jsou obsaženy v souborech se zdrojovými kódy `csindex.h` a `csindex.cpp`. *CsIndex* má základní kódování *KOI8-ČS*. *KOI8-ČS* obsahuje všechna písmena české a slovenské abecedy, písmeno *ch* je kódováno jako jedno písmeno, obsahuje také cyrilici a další základní znaky. *CsIndex* však umožňuje použít i jiná kódování. Toho lze docílit pomocí konverzních tabulek, které je třeba připsat do zdrojového kódu a přeložit C++ kompilátorem. Z dnešního pohledu je kódovaní *KOI8-ČS* zastaralé. Používalo se převážně v osmibitových systémech a i přesto, že toto kódovaní bylo normované ČSN 36 9103, ani v době své největší slávy nebylo příliš používané. Způsob jakým *CsIndex* řeší jiná kódování a abecední řazení, je na dnešní dobu nevyhovující. Při každé změně by bylo potřeba program znova kompilovat.

3.4 SharpIndex

Cílem této bakalářské práce je navrhnout a vytvořit nový indexovací program. Tento program ponese jméno *SharpIndex*. *SharpIndex* bude vytvářet soubor `.ind` ze souboru generovaného LaTeXem a použije vhodné funkce a rozdělení, které se osvědčily v programech *MakeIndex* 3.1 a *Xindy* 3.2. Program bude vytvořený v jazyce C# s použitím technologie *.NET* a bude ovládaný přes příkazovou řádku. Uživatelé LaTeXu jsou zvyklí pracovat v textovém režimu, proto grafické uživatelské rozhraní není nutné. Dále bude podporovat kódování *UTF-8* v rozsahu pro pokrytí všech znaků české abecedy podle normy. Předpokládá se však možnost použít i jiná kódování jako například *ANSII* nebo *Windows-1250*. Abecední pořadí se může v různých jazycích lišit. Musí tedy existovat možnost toto pořadí změnit, nebo mít možnost vytvořit více různých pravidel pro třídění a poté si jedno z nich vybrat. Takto daná pravidla musí být pro uživatele čitelná, proto je bude možné editovat pomocí externího *XML* souboru. Podrobnější popis bude uveden v kapitole 5.

4 České abecední řazení

Správné třídění slov podle české abecedy určuje norma ČSN 97 6030 z roku 1994 [2]. Tato norma určuje i pořadí slov napsaných v jiném než českém jazyce, pořadí číslic, znamének a obrazců, jak v ručně psané, tak v elektronické formě.

4.1 Definice pojmů

Na začátek je nutné vysvětlit několik pojmů. *Grafém* je jakýkoli písemný znak. *Písmeno* je jakýkoli grafém, který je součástí abecedy. *Abeceda* je množina písmen, jež má danou posloupnost, aneb abecední pořadí. *Mezinárodní latinská abeceda* je soubor písmen, jež obsahuje pouze základní latinská písmena. Tyto písmena neobsahují žádná diakritická znaménka ani spřežky. *Národní abecedou* se rozumí soubor všech písmen, jež se vyskytují v daném jazyce. Patří sem jak písmena s diakritickými znaménky, tak i spřežky. *Standardizovaná národní abeceda* vychází z národní abecedy. Navíc je doplněna o další písmena nebo symboly, které jsou potřebné k řazení cizích nebo přejatých slov. *Standardizovaná národní abeceda* tvoří pevně stanovené pořadí písmen pro daný jazyk či národnost.

Dále je nutné určit, jak čeština definuje slova ve smyslu řazení. *Slovo* je souvislá řada písmen oddělená mezerou. V elektronické formě může být jedno slovo i složenina tvořená z více slov, musí však místo mezery být použita *non-breaking space* (nbsp). Čeština definuje tři úrovně při řazení slov, *heslo*, *podheslo* a *doplněk*. *Heslo* má nejvyšší třídící platnost. *Podheslo* je upřesňující informace k heslu, k níž se při třídění přihlíží pouze pokud jsou totožná hesla. Nejnižší třídící platnost má *doplněk hesla*, podle kterého se třídí pouze v případě shody i v podheslu. V případě použití podhesel a doplňků je nutné se dohodnout na použitých symbolech k oddělení těchto částí slova. Například se dá stanovit, že za pomlčkou následuje podheslo a v závorce je pak doplněk. Slovem tedy může být složenina „Novák, T. - Praha (1987)“. Heslo je „Novák, T.“, podheslo „Praha“ a doplněk „1987“.

Jednotlivá písmena ve slovu se porovnávají podle pozice prvku řazení v abecední sestavě. Prvek řazení může být základní písmeno (a, á, ...), znak ($, &, ...) i speciálně upravený údaj (ch, ss, ...). Uspořádaný soubor prvků řazení se nazývá *abecední řazení*. *Abecední sestava* je abecední řazení doplněné o další informace nutné k setřídění slov. Například doplnění o informaci „Tečka ve smyslu zkrácení slova se řadí jako diakritické znaménko posledního písmene dané zkratky slova.“. Toto pravidlo říká, že zkratka *atd.* má poslední písmeno *d* s tečkou. Nevadí, že písmeno *d* s tečkou neexistuje, o jeho umístění do abecedního řazení se rozhodne dle pravidel zmíněných v odstavci 4.3.

4.2 Porovnávání prvků řazení

Porovnávání slov dle české normy se uskutečňuje postupným porovnáním prvků řazení dle abecední sestavy tvořené *standardizovanou národní abecedou*. Pro češtinu existuje dvou průchodové řazení. Při prvním průchodu se berou v potaz pouze prvky *základního řazení*. V případě shody dvou různých slov tvořenými z písmen české standardizované národní abecedy dochází k druhému průchodu, kdy se rozlišují i prvky řazení se *sekundární třídící*

a	b	c	č	d	e	f	g	h	ch	i	j	k	l	m
n	o	p	q	r	ř	s	š	t	u	v	w	x	y	z

Tabulka 4: Základní prvky řazení dle české standardizované abecedy.

á	ď	é	ě	í	ň	ó	ť	ú	ů	ý

Tabulka 5: Prvky řazení se sekundární třídící platností.

platností, interpunkční znaménka a obrazce. Malá a velká písmena mají stejnou třídící platnost.

4.2.1 Základní prvky řazení

Všechna písmena, která se nevyskytují v seznamu písmen základního řazení 4, se považují za základní prvek řazení bez diakritického znaménka. Písmeno *á* se řadí jako *a*. Pokud se mezi slovy vyskytuje více mezer, při řazení se použije pouze jedna. Mezera se řadí před všechna písmena.

4.2.2 Prvky řazení se sekundární třídící platností

Mezi prvky se sekundární třídící platností patří písmena v tabulce 5 a písmena z jiných abeced jako *ń*, *ö* a další.

4.3 Řazení písmen s diakritickými znaménky

Pokud má základní písmeno diakritické znaménko, řadíme jej podle polohy tohoto znaménka v tomto pořadí. Nejprve se řadí ta písmena, u nichž se diakritické znaménko vyskytuje nahoře, poté ta se znaménkem dole, vzadu, vpředu a nakonec ta s diakritickým znaménkem uvnitř nebo přes. Pokud o pořadí písmene nelze rozhodnout podle polohy znaménka, záleží na počtu čar. Nejvyšší prioritu mají jednoduchá znaménka (viz. tabulka 6), následují dvojitá znaménka (viz. tabulka 7) a nakonec trojitá (viz. tabulka 8).

◌̇	◌̓	◌̄	◌̍	◌́	◌̀	◌̂
dot	accent	macron	vertical stroke	acute	grave	circumflex

◌̌	◌̃	◌̆	◌͡	◌̧	◌̨	◌̊
caron	tilde	breve	tie	cedilla	ogonek	ring

Tabulka 6: Jednoduchá diakritická znaménka a jejich priorita v abecedním řazení.

¨	˝	``
diaresis	double acute	double grave

Tabulka 7: Dvojitá diakritická znaménka a jejich priorita v abecedním řazení.

∴
triple dot

Tabulka 8: Trojitá diakritická znaménka a jejich priorita v abecedním řazení.

4.4 Interpunkční znaménka

V případě, že k rozlišení dvou různých slov nestačí ani diakritická znaménka, záleží na interpunkčních znaménkách. Dle normy existuji čtyři případy. Interpunkční znaménko se vyskytne ve vzorci, značce nebo šifře. V tom případě se řadí podle pořadí grafému a daného interpunkčního znaménka. Příklad takto setříděných vzorců by bylo *a*, *a-*, *a-b*, *a-b-*, *a-c*, *á*, *å*. Druhý případ nastane, pokud se interpunkční znaménko vyskytne jako heslo samo o sobě (například samotný znak tečka). Pak se řadí nejdříve znaménka (viz. tabulka 9), pak ustálené značky (viz. tabulka 10), grafické značky (viz. tabulka 11) a nakonec obrazce (viz. tabulka 12). Grafické značky se navíc řadí na grafické značky tvořené z úseček, z křivek a z úseček a z křivek. Zároveň záleží na počtu čar, kde nekřížící se čáry se řadí před křížící. U obrazců záleží na počtu hran. Třetí možný výskyt interpunkčního znaménka je při podřazování smíšených hesel a to pouze v případě, že mají stejnou písemnou část. Tady určují pořadí řazení znaménka stejná pravidla, jako by se vyskytovalo samostatně. Příkladem může být posloupnost *myš (domácí)*, *myš [domácí]*, *myš {domácí}*. Poslední čtvrtý případ nastane, pokud interpunkční znaménko je součástí slovního výrazu. Příkladem může být slovo *dostane-li*. K takovýmto znaménkům se při základním třídění nepřihlíží a řadí se jako *dostaneli*.

4.5 Doplňující pravidla

Apostrof a tečka ve významu zkrácení slova, se sekundárně řadí jako písmeno, před kterým se vyskytuje a řídí se pravidly pro diakritická znaménka. Stejná pravidla platí i pro indexy a exponenty. Pokud je dané slovo napsané v cizím písmu (azbuka, cyrilice), je možné použít transliteraci. Například slovo *CYBOPOB*, lze řadit jako *suvorov*, α jako *alfa*, *ß* jako *ss*, *æ* jako *ae*. V případě, že se transliterace nepoužije, dané slovo se řadí až za abecedu a číslice. U řazení čísel rozhoduje jejich numerická hodnota. Přihlíží se ke znaménku a desetinné čárce. Čísla tedy můžeme setřídit do posloupnosti -9.8, -0.8, 0, 0.8, 4, 4.7. Česká norma netřídí římské číslice. Před začátkem třídění je tedy zapotřebí přepsat všechny římské číslice na arabské.

.	,	;	?	!	:	”	-	|	/	\	//	[	]	<	>	{	}	\\

Tabulka 9: Pořadí interpunkčních znamének.

&	£	§	%	‰	$

Tabulka 10: Pořadí ustálených značek.

=	+	×	∗	#	∼	≈	≅

Tabulka 11: Pořadí grafických značek.

4.6 Omezení při strojovém zpracování rejstříků

Česká norma stanovující české abecední řazení [2], obsahuje definice, o kterých nelze strojově rozhodnout. Například celá část o interpunkčních znaménkách 4.4. Nelze rozhodnout, zda interpunkční znaménko je součástí šifry nebo vzorce. Pokud by se uvnitř příkazu `\indexentry` vyskytlo heslo `Morseova abeceda!T-`, pomlčka je zde jako součástí hesla. V hesle `Základní operace!+-*/` by však jako součástí šifry nebo hesla nebyly. Problém nastává i v případě řazení grafických značek. Norma říká, jak se grafické značky mají řadit, existuje příklad těchto značek, není už pak ale řečeno, jak je grafická značka definovaná a neexistuje seznam grafických značek s jejich pořadím. Podřazování hesel pomocí interpunkčních znamének by bylo možné naimplementovat v případě, že by se předem definovala znaménka, která budou použita k podřazení. Toto však nedává moc smysl, neboť LaTeX pro podřazování používá znak „!“ a není důvod, proč tuto vestavěnou funkcionalitu nepoužít. Dále nelze strojově rozhodnout o tom, zda interpunkční znaménko je součástí slovního výrazu či ne. V heslu *dostane-li se* je součástí slovního výrazu a slovo by se tedy mělo řadit jako *dostaneli se*. V heslu *malý-větší-největší* však pomlčka součástí slovního výrazu není, a slovo by se mělo řadit jako *malý-větší-největší*.

Co se týče sekce o doplňujících pravidlech 4.5, lze ji splnit jen z části. Nelze strojově rozhodnout, zda tečka je nebo není ve významu zkrácení slova (*atd.*, *např.* je ve významu zkratky, *ahoj.*, *x.y* není). U čísel má záležet na znaménku a desetinné čárce a čísla se mají řadit vzestupně. Problém je v tom, že existuje velké množství zápisů čísel, různé soustavy, formáty, použité symboly pro desetinnou čárku atd. Tento problém je řešitelný, ale byl by komplikovaný a zbytečně by zpomaloval výsledný třídící algoritmus. Poslední nerozhodnutelný problém je rozpoznání římských číslic a jejich převod na číslice arabské (*V* převést na *5* a *V sobotu* převést na *5 sobotu*?).

Všechna ostatní pravidla v sekcích 4.2, 4.3 a 4.5 budou zohledněná a zakomponovaná do programu *SharpIndex* tak, ať odpovídá normě ČSN 97 6030 z roku 1994 [2].

△	▽	□	◇	★	○

Tabulka 12: Pořadí obrazců.

5 Implementace indexovacího nástroje SharpIndex

Při implementaci indexovacího nástroje *SharpIndex* byl kladen důraz snadnou použitelnost, jednoduchou editaci třídících pravidel a nalezení způsobu, jak odhalit nepřesnosti v třídění a běhu programu samotného. Program má pouze tři argumenty pro příkazovou řádku, veškerá pravidla jsou nastavována v jednom konfiguračním souboru a pro kontrolu je zde možnost zápisu do logu, který obsahuje jak načtená data, použitá pravidla při třídění, tak i výpis informačních a chybových zpráv. Pro implementaci bylo vybráno vývojové prostředí Microsoft Visual Studio 2013 ve verzi 12.0.40629.00 Update 5.

5.1 Konfigurační XML soubor

Většina nastavení, včetně převodů velkých a malých písmen, LaTeX příkazů, abecedního řazení a transliterace se provádí pomocí konfiguračního *XML* souboru. Výchozí konfigurace pro české abecední řazení s kódováním *UTF-8* se nachází v souboru s `utf-8.xml`. Všechny znaky, jejich kódy a názvy, jsou převzaty z Unicode 8.0 Character Code Charts [3]. Konkrétně tedy *Basic Latin (ASCII)* (znaky 0000–007F), *Latin-1 Supplement* (znaky 0080–00FF), *Latin Extended-A* (znaky 0100–017F). Tyto tři sady v sobě obsahují všechny znaky jež obsahuje česká standardizovaná národní abeceda. Pro konfigurační soubor je vytvořený soubor *XML Schema Definition*, `config.xsd`, pomocí kterého je kontrolována struktura dokumentu, datové typy a tím usnadňuje jeho editaci. Kořenem je element *config*, ve kterém je odkaz na *XML* schéma. Kořen obsahuje pět hlavních elementů, a to *chart*, *tex*, *letterSize*, *transliteration* a *order*.

Chart slouží pro kontrolu znaků. Nastavuje se zde množina znaků, jež se očekává při vstupu. Pokud během běhu programu bude načtený znak, který do této množiny nepatří, program bude ukončen s vyjímkou ItemException. Tato funkcionalita je zde z toho důvodu, že pokud by byl načten znak, jež nepatří do abecedy, abecední řazení by nemohlo být seřazeno správně. Množina očekávaných znaků se dá nastavit elementem *range*, u kterého je vnitřní text dle vzoru `([0-9])+(-)([0-9])+`, tedy interval včetně zadaných dekadických čísel, například 11-383. Kromě elementu *range* lze použít i *list*, kde vnitřní text jsou dekadické kódy znaků oddělené čárkou. Element *list* používá vzor `([0-9])+((,)([0-9])+)*`, konkrétně může tedy nabývat hodnot 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, … nebo i jediné hodnoty, například 120. K vyjádření množiny lze použít buď jen jeden z elementů nebo i oba současně.

Tex má za úkol převádět příkazy z typografického systému LaTeX. LaTeX dovoluje psát dokumenty v čité *ASCII* formě i při výskytu znaků které se v této sadě nevyskytují. Toho je docíleno pomocí příkazů, jako je například `\'{a}`, který vypíše znak *á*. Příkaz `\'` tedy přidá nad písmeno čárku (acute), které je uvnitř parametru. Problém je v tom, že LaTeX dovoluje psát příkazy, které dělají to samé, různými způsoby. Příkaz `\'{a}` tedy lze zapsat jako `\'a`, `{\'a}` nebo `\'a{}`. Proto, ať je usnadněno parsrování těchto příkazů, atribut *command* uvnitř `config/tex/conversion` elementu musí být ve formátu `\příkaz{parametr}`. V případě, že daný příkaz parametr nemá, musí u něj být prázdné složené závorky `{}`, jako je tomu například u polského přeškrtnutého *ł*, které napíšeme pomocí `\l{}`. Element *conversion* obsahuje ještě další dva atributy, *convertTo* a

desc. Atribut *convertTo* specifikuje, na jaký řetězec se má příkaz převést, jeho délka není omezená. Atribut *desc* slouží pouze jako popis daného převodu. Element *conversion* není povinný a počet výskytů není omezený. Všechny zmíněné atributy tohoto elementu však povinné jsou.

LetterSize slouží k převodu velkých a malých písmen. Je zde povolen pouze element jednoho typu, jeho počet ale není omezený. Název tohoto elementu je *letter*. *Letter* má tři povinné atributy *upper*, *lower* a *desc*. *Upper* i *lower* jsou datového typu string a jejich hodnoty odpovídají velkému a malému písmenu daného písmene v elementu *letter*. Pro *upper* a *lower* je použit hexadecimální zápis znaků, například znak *a* je zapsaný jako `A`. Je možné použít i klasický zápis znaků, písmeno *a* zapsat jako `a`. Atribut *desc* slouží pro popis daného písmene. Tento popis slouží převážně pro přehlednost při editaci konfiguračního souboru, k chybovým hlášením programu či výpisu aktuální konfigurace programu.

Transliteration slouží, jak už název napovídá, k transliteraci. *Transliteration* může obsahovat pouze jeden element, element *literal*, jehož počet není omezený. *Literal* má tři povinné atributy, *ncr*, *convertTo* a *desc*. *Ncr* je zkratka pro *Numeric Character References*, kterou používá The Unicode Consortium [3] pro obecné označení znaků. *ConvertTo* je atribut, který určuje na jaký řetězec se má *ncr* přepsat. *Ncr* i *convertTo* je datového typu string, kde je doporučené použít hexadecimální zápis znaků. Atribut *desc* slouží ke stejnému účelu jako *desc* u elementu *letterSize*. *Transliteration* se používá pro přepis písmen tak, ať jsou zařaditelné do české abecedy. Například německé písmeno *ß* se podle normy řadí jako *ss*. *Transliteration* je možné použít pro transliteraci jiných abeced (cyrilice, azbuka).

Order je část konfiguračního souboru, ve které je určené pořadí grafémů dané abecedy. Jelikož české abecední řazení je dvoufázové, musí existovat element určující základní třídící platnost a element určující sekundární třídící platnost. Element *primary* určuje základní třídící platnost a má dva atributy. Pokud se dále v tomto textu vyskytne výraz *primary order* nebo *primární řazení*, je tím myšlena základní třídící platnost. Atribut *ncr* (Numeric Character References), je znak nebo řetězec s primárním řazením a atribut *desc*, slouží k popisu grafému. Pořadí elementů *primary* v konfiguračním souboru určují primární pořadí v abecedě. Uvnitř elementu *primary* může být element *secondary*. *Secondary* má stejné atributy se stejným významem jako *primary*, ale definuje sekundární třídící platnost. Pokud v elementu *primary* je písmeno *e* a uvnitř jsou dva elementy *secondary* s písmeny *é* a *ě*, znamená to, že písmena *é* a *ě* se podle primárního řazení řadí jako *e* a podle sekundárního řazení jako *é* a *ě*. U sekundárních elementů záleží na pořadí stejně jako u primárních elementů. Element s nižším číslem řádku má vyšší prioritu.

5.2 Hlavní spouštěcí třída

Hlavní spouštěcí třída programu je SharpIndex. Jejím hlavním úkolem je spouštět větší logické celky, přeposílat data, zachytávat vyjímky, vypisovat informace do konzole a logu a v případě chyby program ukončit. Větší logické celku jsou načtení argumentů z příkazové řádky (hlavní spouštěcí metoda Main(string[] args)), samotné spuštění tvorby rejstříku (metoda run(string configPath, string inputPath, string outputPath)), načtení konfiguračního souboru (metoda loadConfig(string path)), načtení vstupního souboru

(metoda loadIndex(string path)), zápis rejstříku do souboru a zápis logu do souboru (metody writeIndex(string path, ItemCollection itemCollection) a writeLog()). Hlavní třída také obsluhuje vyjímky v případě chyby pomocí metody handleException(Exception e).

5.3 Argumenty příkazové řádky

Program se pod OS Windows spouští pomocí Commmand Line s dvěma povinnými a jedním volitelným parametrem. Povinnými jsou parametry `-config [configPath]` určující cestu ke konfiguračnímu `.xml` souboru a `-input[inputPath]` určující cestu ke vstupnímu `.idx` souboru. Nepovinným parametrem je `-log`, který udělá to, že před skončením programu zapíše záznam o chybových a informačních zprávách, načtených datech a průběhu zpracování rejstříku do souboru `sharpindex.log`. Tento log je zvlášť užitečný v případě, že rejstřík se zdá být vysázený nepřesně. To je nejčastěji způsobeno nesprávným nastavením konfiguračního souboru. Může se však stát, že rejstřík je vysázený správně, ale správnost řazení není na první pohled zřejmá. Log obsahuje podrobné informace o klíčových heslech, primárním a sekundárním řazení.

O načtení argumentů z příkazové řádky se stará třída Arguments. Jejím úkolem je načíst proměnné *configPath*, *inputPath*, *outputPath* a *log*. *ConfigPath* je hodnota předaná parametrem -config, *inputPath* je hodnota předaná parametrem -input. Cesta k vstupnímu souboru musí mít příponu `.idx`. Z této cesty se pak odvozuje cesta k výstupnímu souboru *outputPath*, která bude mít jen změněnou příponu na `.ind`. Default hodnota pro *log* je false, je však doporučené používat zápis do logu vždy. V případě, že je načtený neznámý argument, chybí povinný argument nebo argument je v nesprávném formátu, je vyhozena vyjímka ArgumentException. Po úspěšném načtení argumentů se spustí hlavní část programu, jež má za úkol vygenerovat rejstřík.

5.4 Načtení konfiguračního souboru

Každému hlavnímu elementu z konfiguračního souboru odpovídá třída se stejným názvem. Konkrétně tedy Chart, Tex, LetterSize, Order a Transliteration. Jednotlivé třídy se inicializují příslušnou metodou z třídy Config, která vrací požadovaný typ objektu. Všechny tyto třídy lze zapsat do logu, protože implementují rozhraní IWritable 1. Třídy LetterSize, Order a Transliteration navíc mají podobnou vlastnost, že klíčem jejich dat musí být jeden nebo dva znaky. V české standardizované abecedě a snad ani v cizích abecedách neexistuje písmeno nebo literál, jež by byl tvořený třemi grafémy a přitom měl zvláštní třídící význam jednoho písmene. Toho lze využít k rychlejšímu třídění a vyhledávání, neboť klíčem pak může být char a ne string. V případě jednoho znaku se uloží data do slovníku Dictionary<char, T> a v případě dvou znaků do slovníku Dictionary<char, Dictionary<char, T>>. Tyto tři třídy tedy implementují rozhraní IDataStructure<T> 2 a může k nim být přistupováno stejným způsobem.

```
namespace SharpIndex
{
    public interface IWritable
    {
        string Write();
    }
}
```

Výpis 1: Rozhraní IWritable.

```
using System;

namespace SharpIndex
{
    public interface IDataStructure<T>
    {
        void AddOne(char character, char[] charArray, String description);

        T GetOne(char character);

        void AddTwo(char firstCharacter, char secondCharacter, char[] charArray, String
            description);

        T GetTwo(char firstCharacter, char secondCharacter);
    }
}
```

Výpis 2: Rozhraní IDataStructure.

```
private void addIntoDataObject<T,U>(ref T dataObject, string str, char[] charArray, string desc,
    string xPath)
    where T : IDataStructure<U>
{
    if (str.Length == 1)
    {
        dataObject.AddOne(str[0], charArray, desc);
    }
    else
    {
        if (str.Length == 2)
        {
            dataObject.AddTwo(str[0], str[1], charArray, desc);
        }
        else
        {
            throw new ConfigException(messageAttributeWrongLength(str, xPath));
        }
    }
}
```

Výpis 3: Metoda addIntoDataObject.

Konfigurační soubor je načítaný pomocí XPathNavigator, uvedený v *.NET 4.0* [4], do kterého je načten kořen, element *config*. Třída obsahuje příkazy XPath ke všem načítaným elementům. Skupina požadovaných elementů se vybere konkrétním XPath příkazem a je načtena do XPathIteratoru, který dovoluje těmito uzly procházet. Jednotlivé atributy vybraného uzlu jsou načítány metodou getAttributeValue(XPathNavigator attribute, string expectedName, string xPath), která vrátí textovou hodnotu atributu v XPathNavigatoru. V případě, že se jedná o neočekávaný atribut, vyhodí vyjímku ConfigException. Po úspěšném načtení atributů jednoho elementu jsou data předána příslušnému objektu, který data uloží do své datové struktury. Pokud objekt implementuje rozhraní IDataStructure<T> 2, je pro předání dat použita metoda addIntoDataObject(...) 3.

5.4.1 Chart.cs

Třída Chart uchovává seznam s povolenými znaky, kontroluje zda slova mají povolené znaky, v případě chyby vyhodí vyjímku ItemException. Během načítání dat z elementů obsažených v `config/chart`, si třída Config ukládá maximum z načtených hodnot. Toto maximum je pak použité při vytvoření pole typu bool, ve kterém pozice reprezentuje kód znaku a hodnota na dané pozici určuje, zda znak je či není součástí abecedy. Zmíněné maximum + 1 je pak použito jako velikost pole. Ačkoli největší možný kód znaku v *UTF-8* může být až 0x10FFFF, v Chart je nejvyšší povolený kód 0xFFFF, proto ať se vleze do datového typu char. Rozmezí 0–65535 pro českou standardizovanou abecedu je plně dostačující. Třída Chart.cs používá dvě metody pro přidání dat. Metoda AddRange(List<int[]> range) přidá intervaly s povolenými kódy znaků, meze intervalu jsou přitom také povoleny. Metoda AddList(List<int[]> list) přidá seznamy s výčtem povolených prvků. Samotná kontrola znaků probíhá v metodě TestInChart(string sortAs), která nemá návratovou hodnotu, ale v případě nepovoleného znaku vyhodí vyjímku ItemException. Chart obsahuje metodu Write(), která vrací string s kódy všech znaků které jsou povoleny. Tato metoda je implementací rozhraní IWritable 1.

5.4.2 Tex.cs

Třída Tex převádí TEXové příkazy, ať je možné hesla správně třídit. Pokud by se v heslu rejstříku vyskytl příkaz a ten by nebyl převeden, heslo by pak bylo nesprávně setříděno. Slovo *nádraží* by mohlo být zapsáno jako `n\'{a}dra\v{z}\'{i}` a to rozhodně není tvar, podle kterého by se mělo heslo řadit. Třída Tex obsahuje data reprezentované samostatnou třídou TexData. Tyto data jsou ukládaná do slovníku, kterému je klíčem řetězec reprezentující TEXový příkaz. TexData má v sobě pouze dvě proměnné, které se nastaví v konstruktoru a poté slouží jen ke čtení. Jedná se o *convertTo*, určující na jaký řetězec se má příkaz převést a *description*, který slouží jako popis příkazu při výpisu do logovacího souboru. Příkazy se přidávají pomocí AddTexCommand(string command, string convertTo, string desc).

Hlavním úkolem této třídy je samotný převod hesla. Ten je realizovaný metodou ConvertTexCommands(string word), která vyhledá TEXové příkazy v parametru *word* a převede je. Jelikož příkazy musí být ve formátu `\příkaz{parametr}`, při prohledá-

vání hesla se nejprve hledá znak \. Při nalezení tohoto znaku se pomocí metody getCommand(string command) vyhledá příkaz. Po vyhledání příkazu metoda getCurlyBracket(string parameters) vrátí parametr. Výsledný příkaz pak vznikne složením těchto dvou řetězců, který se vyhledá ve slovníku metodou getConvertTo(string command). Tato metoda vrátí ze slovníku podle klíče řetězec, na který se má příkaz převést. Během tohoto procesu může nastat několik vyjímek typu ItemException. Vyjímka bude vyhozena pokud příkazu chybí parametr, pokud parametr není uzavřený nebo pokud nalezený příkaz není ve slovníku. Třída Tex implementuje rozhraní IWritable 1, metoda Write() vrátí textovou reprezentaci dat pro zápis do logu.

5.4.3 LetterSize.cs

Třída LetterSize uchovává data pro převod velkých písmen na malá. Tato data jsou tvořená třídou LetterSizeData. LetterSizeData má dvě proměnné, jež se nastavují v konstruktoru a poté jsou pouze ke čtení. První je proměnná *letter*, ve které je uloženo odpovídající malé písmeno. Druhou proměnnou je *desc*, která slouží pouze pro popis do logu. LetterSizeData navíc implementuje rozhraní IConvertTo 4, které se používá u IDataStructure<T> 2 k definování pole znaků, na které se má převádět klíč. LetterSize tedy implementuje metodu GetConvertTo(), co vrací proměnnou *letter*.

LetterSize může mít jako klíč jeden nebo dva znaky, proto implementuje rozhraní IDataStructure<T> 2. Obsahuje dvě metody pro přidávání dat, AddOne(char upper, char[] lower, string desc) a AddTwo(char firstUpper, char secondUpper, char[] lower, string desc). Pro získání dat používá metody GetOne(char letter) a GetTwo(char firstLetter, char secondLetter). Tyto metody mohou vrátit i null, což je v pořádku, protože ne všechny grafémy jsou písmena, takže pokud je návratovou hodnotou null, pro převod se použije samotný klíč. Tato třída implementuje rozhraní IWritable 1. Metoda Write() vrací textovou reprezentaci dat pro zápis do logu.

```
namespace SharpIndex
{
    public interface IConvertTo
    {
        char[] GetConvertTo();
    }
}
```

Výpis 4: Rozhraní IConvertTo.

5.4.4 Transliteration.cs

Třída Transliteration provádí transliteraci a uchovává data k tomu potřebná. Svou funkcionalitou a strukturou je velmi podobná třídě LetterSize 5.4.3. Implementuje rozhraní IDataStructure<T> 2 a IWritable 1. Metody pro vrácení dat můžou také vracet null (ne všechny grafémy se transliterují). Datovou strukturu této třídy je třída TransliterationData. Má dvě proměnné, *convertTo* a *desc*, na jaký řetězec se má transliterovat a popis této

transliterace. TransliterationData implementuje rozhraní IConvertTo 4. Metoda GetConvertTo() vrací proměnnou *convertTo*.

5.4.5 Order.cs

Třída Order shromažďuje data o pořadí grafémů a o primárním a sekundárním třídícím pravidle. Svou funkcionalitou a strukturou se opět velmi podobá třídám LetterSize 5.4.3 a Transliteration 5.4.4. Jako data používá třídu OrderData, která má tři proměnné. Proměnná *primary* určuje, jak se daný grafém bude primárně řadit. Proměnná *priority* určuje prioritu grafému. Čím menší číslo, tím vyšší priorita. Proměnná *desc* slouží jako slovní popis při zápisu do logu. Všechny tři proměnné se nastaví v konstruktoru a poté jsou již jen ke čtení. Metoda GetConvertTo() vrací proměnnou *primary*. Je implementována metoda Write() z rozhraní IWritable 1.

Po každém přidání nových dat do třídy Order, se inkrementuje lokální proměnná *priority*. Žádné dva různé grafémy nemohou mít stejnou prioritu, každý grafém má unikátní prioritu. Rozdíl oproti předchozím implementacím rozhraní IDataStructure<T> 2 je také v tom, že metoda GetOne(char character) nemůže vrátit null. Každý znak musí mít známou prioritu řazení. Pokud by tomu tak nebylo, výsledné řazení by nebylo správné. Z toho důvodu metoda GetOne(char character), v případě, že nenalezne data k požadovanému klíči, vyhodí vyjímku ConfigException. Metoda GetTwo(char firstCharacter, char secondCharacter) může vrátit null, protože ne každé spojení dvou znaků musí mít svou vlastní prioritu.

Data do třídy Order se nenačítají přímo jak v předchozích případech. Jelikož v konfiguračním souboru je element *primary*, uvnitř kterého může být element *secondary* a Order má pouze jeden typ dat OrderData, musí existovat vhodný způsob jak tato data zapsat. Data se přidávají metodami AddOne(char grapheme, char[] primary, String description) a AddTwo(char firstCharacter, char secondCharacter, char[] primary, string desc). V případě, že se přidává element *primary*, bude proměnné *primary* v metodách pro přidávání dat odpovídat *ncr* z elementu *primary*. V případě, že se přidává element *secondary*, bude proměnné *primary* v metodách pro přidávání dat odpovídat *ncr* z elementu předka, tedy elementu *primary*.

5.5 Datová struktura pro rejstřík

Rejstřík, tak jak je definovaný v TEXu, může mít maximální hloubku tři. Pokud jsou shodné hesla v dané hloubce, příslušné podhesla ve větší hloubce se nutné znova setřídit. Pokud se i mezi nimi najde shodné podheslo, postupuje se stejným způsobem dále. S datovou strukturou, která by měla fixně nastavenou hloubku na tři úrovně by se špatně pracovalo. Mnohem lepší by byla struktura, která by se mohla rozvětvovat do libovolné hloubky, i když by byla využívána pouze do hloubky tři. Struktura by se mohla při přidání prvku automaticky třídit a mělo by být možné skrz ni jednoduše iterovat. Tento popis odpovídá struktuře použité v programu *SharpIndex*. Je tvořena dvěma třídami ItemCollection a ItemNode, které tvoří datovou strukturu a třídou WordComparer jež je implementací rozhraní IComparer<T> a stará se o třídění prvků v struktuře.

5.5.1 ItemCollection.cs

Třída ItemCollection má dvě privátní proměnné. Generický slovník *itemNodes* datového typu SortedDictionary<string, ItemNode> a *wordComparer* datového typu WordComparer. ItemCollection se inicializuje konstruktorem, jež má jeden parametr *order*, datového typu IDataStructure<OrderData>, který obsahuje data s pořadím jednotlivých grafémů. Toto pořadí je potřebné k vytvoření compareru, který bude SortedDictionary<T> třídit. K tomu, aby bylo možné hesla třídit dle normy, musí WordComparer ke každému heslu znát primární a sekundární třídící pravidlo. Vytváření těchto pravidel může při delších heslech značně zpomalit běh programu, obzvlášť pokud by se tato pravidla měla generovat pokaždé, když by se ve struktuře porovnávalo. Z tohoto důvodu je nutné před přidáním nového hesla metodou AddItemNode(string key, ItemNode itemNode), přidat primární a sekundární třídící pravidlo přidávaného hesla do wordCompareru. K tomuto účelu má ItemCollection metodu AddSortingRules(string key, char[] primary, char[] secondary).

Pro zjištění zda třída ItemCollection obsahuje item s daným klíčem, existuje metoda ContainsNode(string key). V případě, že item existuje, lze jej vrátit pomocí GetItemNode(string key). Proto ať lze kolekcí iterovat, třída ItemCollection implementuje rozhraní IEnumerable<T> [4], kde T je KeyValuePair<string, ItemNode>. Metoda GetEnumerator() vrací jednotlivé položky v *itemNodes*. Posledním rozhraním, které tato třída implementuje je IWritable 1 a jeho metoda Write(). Write() používá printItemCollection(int currentDepth) 5, kde počáteční *currentDepth* je rovna 1. Díky rozhraní IEnumerable<T> lze v cyklu foreach použít klíčové slovo this. Pro každý záznam se pak přidá do StringBuilderu textový popis uzlu a na tento uzel je rekurzivně volaná metoda printItemCollection(int currentDepth) s *currentDepth* o jedna vyšším. Textový popis uzlu vrací metoda appendItemNode(StringBuilder sb, int depth, char[][] rules, KeyValuePair<string, ItemNode> kvp).

```
private string printItemCollection(int currentDepth)
{
    StringBuilder sb = new StringBuilder();
    foreach (KeyValuePair<string, ItemNode> kvp1 in this)
    {
        appendItemNode(sb, currentDepth, wordComparer.GetRules(kvp1.Key), kvp1);

        ItemCollection subItemCollection = kvp1.Value.GetItemCollection();
        sb.Append(subItemCollection.printItemCollection(currentDepth + 1));
    }
    return sb.ToString();
}
```

Výpis 5: Metoda pro výpis ItemCollection.

5.5.2 WordComparer.cs

Třída WordComparer má dvě hlavní privátní proměnné IDataStructure<OrderData> *order*, což je objekt definující pořadí grafémů a Dictionary<string, char[][]> *rules*, což je slovník obsahující pravidla pro množinu klíčů. Třídící pravidla se do WordCompareru přidávají metodou AddRule(string key, char[] primary, char[] secondary). Metoda public int Compare(string key1, string key2) je implementací rozhraní IComparer<T> [4]. Nepříjemným faktem je, že SortedDictionary<T> používá comparer i při metodě ContainsKey(string key), takže i přes to, že jsou pravidla napřed vkládána do compareru a až pak do SortedDictionary<T>, musí se kontrolovat, zda pravidla pro daný klíč existují. Pokud pravidla neexistují, metoda vrací -1. Pokud pravidla existují, porovnají se klíče dle pravidla *primary* a v případě, že jsou si rovny, porovnají se dle pravidla *secondary*. Porovnání těchto pravidel pak probíhá v metodě compareWords(char[] w1, char[] w2) 6. V této metodě je důležité, že pokud je aktuální index na konci slova, nevyhledává se písmeno tvořené dvěma znaky, že vyhledání priority písmene tvořeného dvěma znaky má přednost před jednoznakovým písmenem a že pokud bylo nalezeno písmeno tvořené dvěma znaky, posouvá se aktuální index o dvě, ne o jedna. Poslední metoda WordComapreru je GetRules(string key), která vrací pravidla pro daný klíč, používá při výpisu ItemCollection do logu.

```
private int compareWords(char[] w1, char[] w2)
{
    ...
    while (w1index < w1.Length && w2index < w2.Length)
    {
        if ((w1index + 1) == w1.Length) //konec slova
        {
            w1priority = order.GetOne(w1[w1index]).Priority;
            w1index++;
        }
        else
        {
            orderData = order.GetTwo(w1[w1index], w1[w1index + 1]);
            if (orderData == null)
            {
                w1priority = order.GetOne(w1[w1index]).Priority;
                w1index++;
            }
            else
            {
                w1priority = orderData.Priority;
                w1index += 2;
            }
        }
        ...
    }
    ...
}
```

Výpis 6: Zjištění priority slova.

5.5.3 ItemNode.cs

Třída ItemNode je konkrétní reprezentací položky rejstříku v dané hloubce. V konstruktoru se předávají dva parametry, jak se položka má zobrazovat v rejstříku a pravidla s pořadím grafémů. Tyto pravidla jsou pak použité k inicializaci proměnné typu ItemCollection, která je třídní proměnnou a umožňuje položky do sebe zanořovat. K vrácení této kolekce pak slouží metoda GetItemCollection().

LaTeX definuje tři typy označení stránky, na které se heslo nachází. První typ označení je, pokud se vztahuje pouze k jedné stránce (v rejstříku jako „[heslo], [číslo]“). Další dva případy nastanou pokud se jedná o interval stránek, LaTeXem označovaný jako hyperpage (v rejstříku jako „[heslo], [číslo1] -- [číslo2]“). Indexy jedné stránky jsou ukládány do proměnné List<int> *singlePageIndex*, intervaly stránek jsou ukládané do proměnné List<int[]> *hyperPageIndex*. Pro typ indexu jsou nadefinovány tři konstanty. Pro levý a pravý index se používá INDEX_ HYPERPAGE_LEFT a INDEX_HYPERPAGE_RIGHT, pro index jediné stránky se používá INDEX_SINGLEPAGE. Metoda, která tyto indexy přidává, se nazývá AddIndex(int indexType, int indexValue), kde *indexType* je jedna ze zmíněných konstant a *indexValue* je číslo stránky. Index typu INDEX_SINGLEPAGE lze přidat rovnou, ale u indexů typu hyperpage, je potřeba uložit levou mez do zvláštní proměnné a až při načtení pravé meze uložit celý interval do *hyperPageIndex*. V případě, že budou hyperpage indexy načteny v nesprávném pořadí a nebudou se navzájem uzavírat, bude vyhozena vyjímka ItemException. ItemNode implementuje rozhraní IWritable 1, které používá při výpisu do logu třída ItemCollection.

5.6 Načtení vstupního .idx souboru

Načtení vstupního souboru je volané v hlavní spouštěcí třídě 5.2 a probíhá ve třech krocích. Nejprve je nutné vytvořit datovou strukturu, do které se budou data ukládat. Touto strukturou je ItemCollection 5.5. Poté se vytvoří se ItemParser 5.6.2, který upravuje data načtená ze vstupního souboru tak, ať je lze do kolekce vložit a vytváří pravidla pro třídění. Nakonec je vytvořen objekt typu InputFile 5.6.1, který předává ItemParseru data načtení ze souboru. Takto se načtou data ze vstupního `.idx` souboru do ItemCollection.

5.6.1 InputFile.cs

Třída InputFile načítá data ze vstupního `.idx` souboru. V konstruktoru třídy se předává cesta k vstupnímu souboru a objekt typu ItemParser. Konstruktor načítá soubor řádek po řádku a kontroluje, zda řádek začíná příkazem `\indexentry`. Pokud ne, vyhodí vyjímku InputFileException. Pokud ano, metodou getCurlyBracketContent(string parameters) vrátí parametry příkazu. Další vyjímka může nastat v případě, že počet parametrů příkazu `\indexentry` je jiný než dva. Prvním parametrem je heslo případně i podhesla, kde navíc může být i informace o tom, že index je hyperindex. Druhým parametrem je číslo stránky, tedy index samotný. Příklad použití hyperpage ilustruje tabulka 13. Údaj o tom, že index je typu hyperpage se tedy nevyskytuje v parametru pro index, ale v parametru pro heslo, což může být matoucí. Z prvního parametru se tedy nejdříve zjistí

\indexentry{Ostrava!VŠB|(hyperpage}{9}	\item Ostrava
\indexentry{Ostrava!VŠB|)hyperpage}{17}	\subitem VŠB, \hyperpage{9--17}

Tabulka 13: Hyperpage index v .idx souboru a jeho zpracování v .ind souboru.

typ indexu a uloží do proměnné *indexType*. Druhý parametr označující číslo stránky se převede na číslo a uloží do proměnné *index*.

Zbytek prvního parametru obsahuje informace o heslu, jeho hloubce a zobrazení. Pro oddělení hloubky LaTeX používá znak „!“. Pokud hloubka hesla není od jedné do tří, bude vyhozena vyjímka InputFileException. V každé úrovni hloubky je povolený znak „@“, který určuje, jak se heslo bude řadit a jak se heslo má zobrazit v `.ind` souboru. Pokud se v jedné úrovni hesla vyskytuje znak „@“ více než jednou, je vyhozena vyjímka InputFileException. Hesla jsou nakonec uloženy v poli *sortAndDisplay* typu string, kde na první pozici je jak se heslo řadí a na druhé pozici jak se má heslo zobrazit. Takto načtené proměnné *index*, *indexType* a *sortAndDisplay* se přeposílají ItemParseru 5.6.2, který data dále zpracovává.

5.6.2 ItemParser.cs

Třída ItemParser potřebuje pro svou funkčnost veškeré objekty, jež jsou načítány ve třídě Config 5.4. Konkrétně tedy *chart*, *tex*, *letterSize*, *order* a *transliteration*. Tyto objekty jsou společně s datovou strukturou pro rejstřík *itemCollection* posílány v konstruktoru této třídy. Pro přidání záznamu do *itemCollection* se používají tři metody s názvem AddEntryDepth1, AddEntryDepth2 a AddEntryDepth3, které se liší počtem parametrů, neboť každá z nich přidává heslo jiné hloubky. Ve výpisu 7 je zdrojový kód metody AddEntryDepth3(...), metody AddEntryDepth1(...) a AddEntryDepth2(...) pracují obdobně.

```
public void AddEntryDepth3(string sortAs1, string displayAs1, string sortAs2, string displayAs2,
    string sortAs3, string displayAs3, int indexType, int index)
{
    ItemNode itemNode1 = getItemNode(itemCollection, sortAs1, displayAs1);
    ItemCollection itemCollection2 = itemNode1.GetItemCollection();

    ItemNode itemNode2 = getItemNode(itemCollection2, sortAs2, displayAs2);
    ItemCollection itemCollection3 = itemNode2.GetItemCollection();

    ItemNode itemNode3 = getItemNode(itemCollection3, sortAs3, displayAs3);
    itemNode3.AddIndex(indexType, index);
}
```

Výpis 7: Metoda pro vložení dat hloubky tři do ItemCollection.

ItemCollection se prohledává postupně od kořene a vrací *itemNode* odpovídající požadovaným parametrům *sortAs* a *displayAs*. Jakmile se vyhledá požadovaný uzel, přidá se do něj index daného typu. Objekt *itemNode* vrací metoda getItemNode(ItemCollection currentItemCollection, string sortAs1, string displayAs1). Tato metoda nejdřív prohle-

dává danou kolekci *currentItemCollection*, zda v ní existuje uzel s požadovanými parametry. Pokud takovýto uzel byl nalezen, vrátí jej a pokud nebyl, vytvoří nový uzel dle zadaných parametrů a taky jej vrátí. Problém může nastat v případě, že se má přidat heslo, které se liší pouze v malých a velkých písmenech. V tom případě by byla vyhozena vyjímka ItemException. Důvodem pro vyhození vyjímky je způsob řazení dle české normy, která nerozlišuje malá a velká písmena. Když by se přidaly dvě slova „vsb-tuo“ a „VSB-TUO“, samotnému třídění by to nevadilo, slova jsou si rovny. Nešlo by však rozhodnout, které z těchto slov vypsat do rejstříku, jestli „vsb-tuo“ nebo „VSB-TUO“. Náhodný výběr jedno z těchto hesel také není vhodný. Z toho vyplývá, že klíčem pod kterým má být heslo uloženo v struktuře, nemůže být řetězec podle kterého se má heslo řadit.

Klíč pro heslo vrací metoda GetKey(string sortAs), která má parametr řetězec, jak se má heslo řadit. Ten se musí pomocí metody tex.ConvertTexCommands(sortAs) zbavit TEXových příkazů, zkontrolovat zda všechny znaky jsou povolené a poté vše převést na malá písmena. Takto vytvořený řetězec je klíčem.

Při vkládání nového slova do *itemCollection* je nutné nejdříve vytvořit pravidla pro primární a sekundární třídění. Řetězec pro sekundární třídění *secondary* odpovídá klíči struktury, ve kterém byly všechny potřebné znaky transliterovány. Řetězec pro primární třídění *primary* odpovídá řetězci pro sekundární třídění, ve kterém byly všechny znaky převedeny na znaky s primární třídící platností. Jelikož třídy LetterSize, Order a Translitaration implementují rozhraní IDataStructure<T> 2, je zde použita jediná metoda, která převádí řetězce dle dat typu T. Tato metoda se jmenuje ConvertWord a její zdrojový kód je ve výpise 8.

Metoda ConvertWord pracuje tak, že postupně převádí vstupní řetězec *word* na nový, *convertedWord*. Prochází vstupní řetězec znak po znaku a testuje, zda se v datovém objektu T vyskytuje záznam tvořený dvěma, případně jedním znakem. Nalezené záznamy dat pak přidává do nového slova *convertedWord* pomocí metody GetConvertTo(). Data datového objektu T musí implementovat rozhraní IConvertTo 4. Jakmile jsou známy proměnné *key*, *primary* a *secondary*, je možno přidat data do *itemCollection*, jak ilustruje výpis 9.

5.7 Záznam načtených dat, informačních a chybových výstupů

O záznam informačních a chybových výstupů a záznam načtených dat, se stará třída Log. Tato třída se inicializuje bezparametrickým konstruktorem, který inicializuje proměnné a nastaví defaultní hodnoty. Třída má tři privátní proměnné. Proměnná bool *writeLog* určuje, zda se má log zapsat do souboru. Seznam List<IWritable> *writableObjects* obsahuje objekty, jež mají být vypsány do logu. Tyto objekty musí implementovat rozhraní IWritable 1. Pro záznam informačních a chybových hlášení slouží objekt typu StringBuilder *consoleLog*. Nové objekty se do logu přidávají metodou AddWritableObject(IWritable writableObject). Metoda WriteIntoConsole(string line) zapíše na výstup do konzole obsah proměnné *line* a přidá kopii tohoto řetězce do *consoleLog*.

```
public char[] ConvertWord<T, U>(ref T dataObject, char[] word)
    where T : IDataStructure<U>
    where U : IConvertTo
{
    U dataStructure;
    List<char> convertedWord = new List<char>();
    int i = 0;
    for (; i < word.Length - 1; i++)
    {
        dataStructure = dataObject.GetTwo(word[i], word[i + 1]);
        if (dataStructure != null)
        {
            convertedWord.AddRange(dataStructure.GetConvertTo());
            i++;
        }
        else
        {
            dataStructure = dataObject.GetOne(word[i]);
            if (dataStructure != null)
            {
                convertedWord.AddRange(dataStructure.GetConvertTo());
            }
            else
            {
                convertedWord.Add(word[i]);
            }
        }
    }
    ...
    return convertedWord.ToArray();
}
```

Výpis 8: Metoda pro převod řetězce dle struktury implementující IDataStructure<T>.

```
char[] keyCharArray = GetKey(sortAs);
string key = new string(keyCharArray);
char[] secondary = ConvertWord<Transliteration, TransliterationData>(ref transliteration ,
    keyCharArray);
char[] primary = ConvertWord<Order, OrderData>(ref order, secondary);
currentItemCollection.AddSortingRules(key, primary, secondary);

itemNode = new ItemNode(displayAs, order);
currentItemCollection.AddItemNode(key, itemNode);
```

Výpis 9: Ukázka přidání nového hesla do datové struktury.

Celý obsah logu pak vrací metoda GetLog(). Seznam *writableObjects* se jednoduše projde pomocí foreach a na jednotlivé položky se volá metoda Write(), jejíž návratová hodnota je přidána do objektu *sb* typu StringBuilder. Nakonec se do *sb* přidá *consoleLog* a takto vytvořený záznam metoda vrací.

5.8 Zápis do souboru

Zápis do souboru má na starost třída OutputFile. Pro zápis rejstříku do výstupního `.ind` souboru slouží konstruktor se dvěma parametry *path* a *itemCollection*. Řetězec *path* určuje cestu k souboru a *itemCollection* je datová struktura obsahující samotný setříděný rejstřík 5.5. K vytvoření textu pro zápis do souboru je použitý objekt *sb* typu StringBuilder, do kterého se přidávají potřebné TEXové příkazy a obsah jednotlivých uzlů. Pro vrácení textové reprezentace uzlu v *itemCollection* je použita metoda appendItemNode(StringBuilder sb, ItemNode itemNode, string itemType). Parametr *itemType* je jedna z konstant ITEM, SUBITEM nebo SUBSUBITEM. Takto se vytvoří záznam uzlu, jehož hloubka odpovídá *itemType* a přidají se k němu pomocí metod singlePageIndexToString(List<int> singlePageIndex) a hyperPageToString(List<int[]> hyperPageIndex) stránky, na kterých se dané heslo nachází. Jelikož třída ItemCollection implementuje rozhraní IEnumerable, celá struktura lze jednoduše vypsat pomocí tří vnořených cyklů foreach, jak je ilustrováno ve výpise 10.

```
foreach (KeyValuePair<string, ItemNode> kvp1 in itemCollection)
{
    appendItemNode(sb, kvp1.Value, ITEM);
    ItemCollection subItemCollection = kvp1.Value.GetItemCollection();

    foreach (KeyValuePair<string, ItemNode> kvp2 in subItemCollection)
    {
        appendItemNode(sb, kvp2.Value, SUBITEM);
        ItemCollection subSubItemCollection = kvp2.Value.GetItemCollection();

        foreach (KeyValuePair<string, ItemNode> kvp3 in subSubItemCollection)
        {
            appendItemNode(sb, kvp3.Value, SUBSUBITEM);
        }
    }
}
```

Výpis 10: Zápis datové struktury ItemCollection do souboru.

Třída OutputFile také zapisuje do souboru objekt *log*. K tomuto účelu slouží konstruktor s jedním parametrem *log*. Ten nedělá nic jiného, než že zavolá metodu objektu *log*, GetLog(), převede ji na string a zapíše do souboru, který má konstantní cestu definovanou v LOG_PATH, `sharpindex.log`.

6 Testy

Pro otestování, zda program *SharpIndex* pracuje správně, byly použity dva různé přístupy. K ověření správnosti českého abecedního řazení dle normy, byl využit *unit testing framework*, který je součástí vývojového prostředí, ve kterém byl program napsán. Jako druhý způsob testování byla použita kontrola výstupních souborů při použití vzorových vstupních a výstupních souborů.

6.1 Unit testy

Visual Studio 2013 má prostředí pro testování, tak zvaný *unit testing framework*. V Solution *SharpIndex* jsou dva projekty. V projektu *SharpIndex* je implementace programu samotného a v projektu *CzechSortingTests* se nacházejí unit testy. Pomocí unit testing frameworku byly napsané testy pro kontrolu povolených znaků, převod malých a velkých písmen, převod TEXových příkazů, vytvoření primárního třídícího pravidla, transliterace a třídění samotného.

Správnost funkčnosti třídy Chart je ověřena v testovací třídě ChartUnitTests, která obsahuje pět testů. Každý test má vstupní řetězec ABCDEFGHIJKLMNOPQRSTUVWXYZ, tedy znaky kódované v intervalu 65–90. Pro definici množiny znaků se testuje zápis pomocí intervalu i zápis pomocí výčtu prvků. První dva testy ověřují zápis pomocí intervalu, kdy jednou interval pokrývá množinu 65–90, očekává se test bez vyhozené vyjímky a podruhé pokrývá interval pouze od 65–89 a je očekávaná vyjímka ItemException. Zcela stejná logika je použita u dalších dvou testů, jen s tím rozdílem, že pro zápis množiny znaků je použit výčet prvků. Poslední test kombinuje oba zápisy, kdy navíc je každý typ zápisu použit dva krát. Intervaly jsou 60–70 a 80–90 a výčty prvků 65–70 a 70–79. Očekává se, že metoda bude úspěšně provedena bez vyjímky.

Pro ověření třídy LetterSize je vytvořena testovací třída LetterSizeUnitTests. U všech testů se očekává, že vstupní řetězec bude převeden na malá písmena. Tato třída obsahuje pět testů. První tři testy obsahují seřazené znaky české národní abecedy, je zde tedy i písmeno „ch“. V prvním testu jsou všechna tato písmena velká, očekává se tedy převod, v druhém testu jsou tato písmena malá, očekává se nezměněný řetězec a v třetím případě jsou lichá písmena malá a sudá jsou velká. Převod písmen musí fungovat i u znaků, jež nejsou písmeny. Čtvrtý test má jako vstupní řetězec čísla „1234567890“. Očekává se nezměněný řetězec. Poslední pátý test má za úkol převést speciální znaky. Vstupní řetězec se skládá ze dvou znaků *SS*, jež reprezentují německé velké ostré *s* a literálu *IJ*. Na výstupu se očekává řetězec „ßij“.

Třída TexUnitTests má za úkol ověřit správnost převodu TEXových příkazů, jež má na starost třída Tex. První tři testy očekávají vyhození vyjímky. V prvním případě má vyhození vyjímky způsobit chybějící parametr příkazu. V druhém případě by měla být vyhozena vyjímka, protože parametr příkazu není uzavřený, chybí znak *}*. Třetí test očekává vyjímku z toho důvodu, že načtený příkaz je neznámý aneb není definovaný v konfiguračním souboru. Následující čtyři testy obsahují známé a správně utvořené příkazy, očekává se tedy úspěšný převod řetězců. První tři z těchto testů převádí řetězec, který má příkaz s parametrem na začátku slova, pak na začátku a uvnitř slova a nakonec dva

příkazy uvnitř a jeden na konci slova. Úplně poslední test převádí příkaz, jež má prázdný parametr.

V třídě TransliterationUnitTests se testují transliterace. Třída Transliteration má za úkol převádět písmena z jiných abeced do české národní abecedy tak, ať je možné tyto znaky řadit. Třída obsahuje dva testy na literály. V prvním testu se v testovaném řetězci vyskytuje literál na začátku a v druhém testu uprostřed. Další dva testy jsou na speciální znaky. Zde se tyto speciální znaky vyskytují v prvním testu uprostřed a v druhém testu na začátku a na konci.

Třída PrimaryRuleUnitTests ověřuje správnost vytvořených primárních třídících pravidel. Jak se tato pravidla vytváří je popsáno v kapitole 5.6.2. Tato třída obsahuje sedm testovacích metod. První test obsahuje všechna písmena s diakritickými symboly české standardizované národní abecedy. Na výstupu se očekává řetězec s písmeny, kde diakritické znaménko mají jen čtyři z nich a to *č*, *ř*, *š* a *ž*. Zdrojový kód této testovací metody je ve výpise 11. Další metoda testuje základní písmena bez diakritického znaménka, u které se očekává nezměněný řetězec a zbylé metody testují pravidla pro řetězce „ďábel“, „čáp“, „řek“, „cesta“ a „12345“.

```
[TestMethod]
public void Primary_áčďéěíňóřšťúůýž()
{
    string input = "áčďéěíňóřšťúůýž";
    char[] expected = "ačdeeinořštuuyž".ToCharArray();
    char[] actual = getPrimaryRule(input);

    CollectionAssert.AreEqual(expected, actual, messagePrimaryRuleFailed());
}
```

Výpis 11: Testovací metoda pro vytvoření primárního třídícího pravidla.

Poslední testovací třída SortingUnitTests testuje třídu WordComparer, která se používá k samotnému třídění slov. Vytváření klíče, primárních a sekundárních pravidel, zde odpovídá tomu, jak se vytváří v programu *SharpIndex*. Je zde celkem šestnáct testů. Tři z nich jsou na porovnávání řetězců s číslicemi, kdy jedno slovo obsahuje číslici a druhé ne, pak obě slova obsahují číslici a nakonec se porovnávají jen čísla. Čísla se dle normy mají řadit až za písmena, mezera úplně na začátek a za číslicemi jsou zbylé znaky. Výjimku tvoří jen *non-breaking space*, která ma nejvyšší prioritu. Pro ověření třídění ostatních znaků je zde metoda testující řetězce „1.“ a „1a“. Pro ověření správného setřídění znaku *non-breaking space*, je zde metoda testující řetězce „da capo“ a „Ďábel“, kde v prvním řetězci je použita místo mezery *non-breaking space*. Norma řadí malá a velká písmena stejně. Toto testuje metoda, která porovnává řetězce „lopata“ a „LOpaTA“, očekávaný výstup vracející *wordComparer* je 0, tedy že jsou si rovny. Zbylé testy obsahují typické slova, které se často používají pro testování abecedního řazení.

6.2 Kontrola vstupu a výstupu programu SharpIndex

Pro kontrolu vstupu a výstupu byly použity tři různé vzorové indexové soubory. Soubory `kniha1.idx` a `kniha2.idx` byly zpracovány pomocí programu *MakeIndex* 3.1 a byly k nim vygenerovány příslušné `.ind` soubory. Soubor `kniha3.idx` byl setříděn pomocí programu *Xindy* 3.2. Zmíněné soubory jsou na přiloženém CD v adresáři `files\samples`, konkrétně `kniha1.idx`, `kniha1.ind`, `kniha2.idx`, `kniha2.ind`, `kniha3.idx` a `kniha3.ind`. U všech souborů je použito kódování *ANSI*. Upravené soubory pro program *SharpIndex* včetně příslušných `.ind` souborů jsou v na CD v adresáři `files\sharpindex`.

Soubor `kniha1.idx` obsahuje anglicky psaný text. Nebylo nutné soubor zvláštně upravovat, pouze bylo změněno kódování znaků na *UTF-8*. Tento soubor byl načtený a zpracovaný programem *SharpIndex* pomocí příkazu „sharpindex -input kniha1.idx -config utf-8.xml -log“. Výsledný soubor `kniha1.ind` pak byl totožný s tím, který vygeneroval *MakeIndex*.

Soubor `kniha2.idx` obsahuje klasicky psaný TEXový text, který obsahuje příkazy pro znaky s diakritickými znaménky. V souboru je čistý *ASCI* text. Hned na druhém řádku v souboru je heslo, které se má řadit jako „Pokorny, Jaroslav“, ale do rejstříku se má vypsat „Pokorný, Jaroslav“. Toto je způsob, jak se nejčastěji řešil problém s tím, že *MakeIndex* neumí třídit česky psaný text. Speciální české znaky mu jednoduše nebyly poslány. Jelikož program *SharpIndex* umí řadit české texty, tento soubor byl pozměněn. Pokud se v souboru vyskytovalo heslo, u kterého byl znak „@“ (znak určující jak heslo třídit a jak zobrazit), byla část hesla podle kterého se má třídit, nahrazená částí hesla, která se má vypsat do rejstříku. Takto se budou při tvorbě rejstříku zohledňovat i české znaky. Jelikož soubor `kniha2.idx` obsahuje příkazy, které nejsou ve formátu `\příkaz{parametr}`, byly všechny jinak zapsané TEXové příkazy upraveny a soubor byl uložen s kódováním „UTF-8“. Tento soubor byl načtený a zpracovaný programem *SharpIndex* pomocí příkazu „sharpindex -input kniha2.idx -config utf-8.xml -log“. Výsledný soubor `kniha2.ind` se již od toho, který vygeneroval program *MakeIndex* lišil. Hlavní rozdíl je v tom, že jsou tříděna písmena s diakritickým znaménkem. Hesla začínající například písmenem „Š“, se vyskytují až za všemi hesly začínajícími písmenem „S“.

Poslední soubor již obsahuje hesla zapsaná přímo pomocí znaků. Příkazy jsou použity pouze pro znaky, jež nejsou na české klávesnici. V tomto souboru byly příkazy upraveny do požadovaného formátu, soubor byl uložen s kódováním *UTF-8* a použit jako vstupní soubor pro *SharpIndex* pomocí příkazu „sharpindex -input kniha3.idx -config utf-8.xml -log“. Výsledné soubory se shodují ve smyslu setřídění. Rozdíl oproti souboru vygenerovaného pomocí *Xindy* byl v tom, že *Xindy* do souboru `.ind` vysázel hesla v *ASCII* formátu. *SharpIndex* vysázel přímo české znaky. Toto není problém, jelikož LATEX umí česky psané znaky do rejstříku vysázet.

7 Rozbor výsledné implementace programu SharpIndex

V programu *SharpIndex* je několik míst, které by bylo vhodné lépe propracovat či vylepšit, stejně jako by bylo dobré některé funkcionality přidat. Stávající verze programu je funkční a řadí správně, pro širší použití ale pár věcí chybí.

7.1 Vylepšení programu

Problémem programu *SharpIndex* je, že příkazy ve vstupním souboru musí být ve formátu `\příkaz{parametr}`. TEXmá mnoho příkazů, které parametr mít nemusí, nemají nebo jej není nutné zapisovat. Bylo by dobré najít vhodný způsob, jak tyto příkazy v heslech vyhledávat a hlavně jak tyto příkazy vhodně zapsat do konfiguračního souboru. Stejně jak existují příkazy pro znaky s diakritickými znaménky, existují příkazy i pro speciální symboly. Tyto příkazy jsou typickým příkladem bezparametrických příkazů, například `\texttrademark` LATEX vysází jako „™“. Za bezparametrickými příkazy se navíc často objevují znaky „{} “, které vkládá mezeru. Bez nich by následující text bezprostředně navazoval. Bezparametrický příkaz pak vypadá jako parametrický.

Dále by bylo dobré vylepšit parsrování vstupního souboru. LATEX definuje, že pokud uživatel chce do rejstříku vypsat znak se speciálním významem (např. „!“ - oddělovač hloubky), je potřeba před tento znak zapsat „"“. Znak zapsaný jako „"!“, nebude mít funkci oddělovače hloubky, ale bude zobrazen jako znak „!“, který bude součástí hesla. *Sharpindex* takovéto uvozování speciálních znaků nepodporuje.

7.2 Rozšíření programu

Konfigurační `.xml` soubor má pro snadnější editaci vytvořené Xml Schema Definition. Toto schéma kontroluje názvy atributů, elementů, zda mají správné datové typy, zda jsou na správném místě. Co ale schéma nekontroluje je to, zda nechybí informace o nějakém grafému. Například pokud mezi elementy *letterSize* bude převod písmene na malé „r“ a písmeno „r“ nebude v elementu *order*, je to jasná chyba v konfiguračním souboru, která v aktuální verzi programu není podchycená a musela by se řešit programově. Podobných příkladů by se dalo najít více.

Konfigurační soubor by mohl navíc obsahovat element *settings* pro nastavení. Zde by se nastavovalo, zda se má před tříděním převádět písmena na malá, jaké kódování souboru bude použito, zda použít primární a sekundární třídění či nikoli nebo jestli použít pro číslování stránek číslice arabské nebo římské.

Z důvodu přenositelnosti by bylo dobré přidat možnost převodu znaků s diakritickými znaménky v heslu v `.ind` souboru na čistý *ASCII* text. To, jestli LATEX písmena s diakritickými znaménky rozpozná nebo ne, záleží na použitém balíčku. Pro češtinu je to `\usepackage[czech]{babel}`. Tyto balíčky ale nejsou povinné a proto je čistý *ASCII* text lepší.

SharpIndex nepřidává do `.ind` souboru příkazy `\indexspace`. Tyto příkazy se používají pro oddělení hesel začínajícími stejným písmenem v hloubce jedna od ostatních.

Rejstřík pak je přehlednější. Konfigurační soubor by měl obsahovat informace o tom, jaké grafémy způsobují toto oddělení a pak je použít při zápisu do `.ind` souboru.

Jako poslední by bylo dobré rozšířit konfigurační soubory. Mohlo by existovat více konfiguračních souborů, které by odpovídaly řazení používanému v různých zemích (slovenské, německé, polské atd.). Bylo by dobré rozšířit abecedu v konfiguračním souboru o další znaky, hlavně o ruskou azbuku. V případě rozšíření o azbuku by element *transliteration* měl obsahovat informace k transliteraci azbuky.

8 Závěr

Cílem této práce bylo analyzovat současnou situaci s tvorbou česky psaných rejstříků pro LaTeX a vytvořit nový program, který by takto psané rejstříky pomáhal vytvářet. Vznikl program *SharpIndex*, který pracuje s kódováním *UTF-8* a řadí hesla podle normy pro české abecední řazení. Inspirací k vytvoření tohoto programu byl především program *Xindy*. *SharpIndex* používá příkazovou řádku a je jednoduchý na použití. Lehce se nastavují třídící pravidla a abeceda pomocí *XML* konfiguračního souboru. Bylo by dobré program rozšířit o převod znaků ve výstupním souboru tak, aby byl celý v čisté *ASCII* formě. Dále by bylo vhodné upravit způsob detekce a zápisu TeXových příkazů v konfiguračním souboru, ať není nutné vstupní soubory upravovat. Celý program je napsaný v moderním jazyce C#, u kterého je i programátorská dokumentace, která tvoří přílohu této bakalářské práce. Program prošel *unit* testy i kontrolou celých výstupních souborů. Úplně na závěr bych rád poděkoval panu docentu Jiřímu Dvorskému, za uvedení do světa LaTeXu a hlavně za možnost vypracovat bakalářskou práci na téma, které má praktické využití.

Tomáš Rapi

9 Reference

[1] Kopka H., Daly P.W. *LaTeX kompletní průvodce*, Česká republika: Computer Press, 2004.

[2] *ČSN 97 6030 - Abecední řazení*, Praha: Český normalizační institut, 1994.

[3] *Unicode 8.0 Character Code Charts*, The Unicode Consortium. Unicode Inc., Jan 1991. Web. 5 July 2015.

[4] *Microsoft API and reference catalog*, Microsoft Developer Network, Microsoft, Apr 1975. Web. 5 July 2015.

A Příloha na CD

Na CD je programátorská dokumentace v souboru `dokumentace.pdf`, která je shodná s dokumentací v příloze B, ale ve verzi vygenerované v programu *Doxygen* fungují hypertextové odkazy a lépe se v ní orientuje. V adresáři `sharpindex\release` se nachází vytvořený `.exe` soubor programu *SharpIndex* spolu s konfiguračním souborem `utf-8.xml` a *XML* schématem `config.xsd`. V adresáři `sharpindex\solution` je solution pro Microsoft Visual Studio 2013, ve kterém byl program vytvořen. V adresáři `files\samples` se nachází vstupní soubory pro programy *MakeIndex* a *Xindy*, spolu s `.ind` soubory, které tyto programy vygenerovaly. V adresáři `files\sharpindex` jsou upravené vstupní soubory pro *SharpIndex* včetně `.ind` a `.log` souborů, které *SharpIndex* vygeneroval.

B Programátorská dokumentace

Na následujících stránkách je příloha k této bakalářské práci, dokumentace k programu *SharpIndex* vygenerovaná pomocí programu *Doxygen*. Tato dokumentace má 62 stran.

SharpIndex

Generated by Doxygen 1.8.10

Mon Jul 27 2015 19:49:03

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 SharpIndex Namespace Reference

Classes

- class Arguments

  třída má za úkol načíst argumenty z příkazové řádky, vrátit je a vyhodit vyjímku v případě chyby
- class ArgumentsException

  vlastní vyjíka která může nastat při načítání argumentů z příkazové řádky
- class Chart

  třída uchovává seznam s povolenými znaky, kontroluje zda slova mají povolené znaky, v případě chyby vyhodí vyjímku
- class Config

  načte konfigurační xml soubor, vrátí jednotlivé moduly, vyhodí vyjímku v případě chyby
- class ConfigException

  vlastní vyjíka která může nastat při načítání konfiguračního xml souboru
- interface IConvertTo

  rozhraní které se používá u IDataStructure k definování pole znaků na které se má převádět
- interface IDataStructure

  generické rozhraní pro datové objekty, které pouzívají jako klíč 1 nebo 2 znaky
- class InputFile

  načítá vstupní .idx soubor a přeposílá načtená data objektu ItemParser, v případě chyby vyhodí vyjímku InputFile↩ Exception
- class InputFileException

  vlastní vyjíka která může nastat při načítání vstupního .idx souboru
- class ItemCollection

  třída reprezentující kolekci objektů ItemNode, které jsou tříděné pomoci WordComparer, proto ať lze procházet touto kolekcí pomocí foreach, implementuje generické rozhraní IEnumerable
- class ItemException

  vlastní vyjíka která může nastat při načítání dat do ItemCollection nebo při třídění ItemCollection
- class ItemNode

  třída reprezentující položku v rejstříku včetně podpoložek
- class ItemParser

  třída se stará o nastavení třídících pravidel a správné vložení dat do ItemCollection
- interface IWritable

  rozhraní které používá třída Log k výpisu dat
- class LetterSize

  třída která uchovává data pro převod velkých písmen na malá
- class LetterSizeData

data o převodu velkého písmene na malé

- class Log

  třída shromazďuje data která se mají zapsat do logu
- class Order

  třída shromažďující data o pořadí grafémů a primárním třídícím pravidle
- class OrderData

  data o pořadí a třídění grafému
- class OutputFile

  třída starající se o zápis dat do souboru
- class SharpIndex

  hlavní spouštěcí třída programu sharpindex, řídí proces vytváření rejstříku, zachytává případné vyjímky
- class Tex

  třída převádí TeXové příkazy ať je možné hesla správně třídit
- class TexData

  třída obsahující data potřebná pro převod TeXových příkazů
- class Transliteration

  třída provádí transliteraci, napr. nemecke ostre "s" na "ss", uchovává data k tomu potřebná
- class TransliterationData

  data o transliteraci
- class WordComparer

  třída se stará o třídění prvků dle nastavených primárních a sekundárních pravidel

Chapter 5

Class Documentation

5.1 SharpIndex.Arguments Class Reference

třída má za úkol načíst argumenty z příkazové řádky, vrátit je a vyhodit vyjímku v případě chyby

Public Member Functions

- Arguments ()

  bezparametrický konstruktor třídy Arguments, nastaví privátní proměnné na počáteční hodnoty
- void Load (string[ ] args)

  načte parametry z příkazové řádky

Properties

- string InputPath `[get]`

  vrátí inputPath, cestu k vstupnímu .idx souboru
- string ConfigPath `[get]`

  vrátí configPath, cestu k vstupnímu .xml souboru
- string OutputPath `[get]`

  vrátí outputPath, cestu k výstupnímu .ind souboru
- bool Log `[get]`

  vrátí zda se má či nemá zapsat logovací soubor

5.1.1 Detailed Description

třída má za úkol načíst argumenty z příkazové řádky, vrátit je a vyhodit vyjímku v případě chyby

5.1.2 Constructor & Destructor Documentation

5.1.2.1 SharpIndex.Arguments.Arguments ( )

bezparametrický konstruktor třídy Arguments, nastaví privátní proměnné na počáteční hodnoty

5.1.3 Member Function Documentation

5.1.3.1 void SharpIndex.Arguments.Load (string[ ] *args*)

načte parametry z příkazové řádky

Parameters

args	parametry příkazové řádky

Exceptions

ArgumentsException	pokud chybí argument -input či -config nebo u těchto argumentů není hodnota, pokud inputPath nemá příponu .idx, pokud se vyskytne neznámý argument

5.1.4 Property Documentation

5.1.4.1 string SharpIndex.Arguments.ConfigPath [get]

vrátí configPath, cestu k vstupnímu .xml souboru

5.1.4.2 string SharpIndex.Arguments.InputPath [get]

vrátí inputPath, cestu k vstupnímu .idx souboru

5.1.4.3 bool SharpIndex.Arguments.Log [get]

vrátí zda se má či nemá zapsat logovací soubor

5.1.4.4 string SharpIndex.Arguments.OutputPath [get]

vrátí outputPath, cestu k výstupnímu .ind souboru

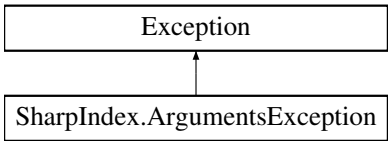
The documentation for this class was generated from the following file:

- C:/doxygen/Arguments.cs

5.2 SharpIndex.ArgumentsException Class Reference

vlastní vyjíka která může nastat při načítání argumentů z příkazové řádky

Inheritance diagram for SharpIndex.ArgumentsException:



Public Member Functions

- ArgumentsException ()

  bezparametrický konstruktor vyjímky
- ArgumentsException (string message)

  konstruktor vyjímky se zprávou
- ArgumentsException (string message, Exception inner)

  konstruktor vyjímky se zprávou a předcházenící vyjímkou

5.2.1 Detailed Description

vlastní vyjíka která může nastat při načítání argumentů z příkazové řádky

5.2.2 Constructor & Destructor Documentation

5.2.2.1 SharpIndex.ArgumentsException.ArgumentsException ( )

bezparametrický konstruktor vyjímky

5.2.2.2 SharpIndex.ArgumentsException.ArgumentsException (string *message*)

konstruktor vyjímky se zprávou

Parameters

message	zpráva o vyjímce

5.2.2.3 SharpIndex.ArgumentsException.ArgumentsException (string *message,* Exception *inner*)

konstruktor vyjímky se zprávou a předcházející vyjímkou

Parameters

message	zpráva o vyjímce
inner	předcházející vyjímka

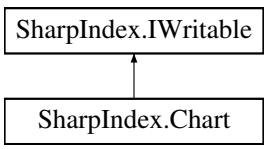
The documentation for this class was generated from the following file:

- C:/doxygen/ArgumentsException.cs

5.3 SharpIndex.Chart Class Reference

třída uchovává seznam s povolenými znaky, kontroluje zda slova mají povolené znaky, v případě chyby vyhodí vyjímku

Inheritance diagram for SharpIndex.Chart:



Public Member Functions

- Chart (int size)

  vytvori objekt chart a inicializuje pole ve kterem jsou povolene znaky abecedy
- void AddRange (List< int[ ]> range)

  přidá intervaly s povolenými kódy znaků, meze intervalu jsou povoleny
- void AddList (List< int[ ]> list)

  přidá seznamy prvků jež jsou povolené
- void TestInChart (string sortAs)

testuje zda zadaný řetězec obsahuje pouze povolené znaky

- string Write ()

  implementace rozhraní IWritable, vytvoří z dat v Chart řetězec pro výpis

5.3.1 Detailed Description

třída uchovává seznam s povolenými znaky, kontroluje zda slova mají povolené znaky, v případě chyby vyhodí vyjímku

5.3.2 Constructor & Destructor Documentation

5.3.2.1 SharpIndex.Chart.Chart (int *size*)

vytvori objekt chart a inicializuje pole ve kterem jsou povolene znaky abecedy

Parameters

size	maximální kód povoleného znaku

5.3.3 Member Function Documentation

5.3.3.1 void SharpIndex.Chart.AddList (List< int[ ]> *list*)

přidá seznamy prvků jež jsou povolené

Parameters

list	List se seznamy povolených prvků

5.3.3.2 void SharpIndex.Chart.AddRange (List< int[ ]> *range*)

přidá intervaly s povolenými kódy znaků, meze intervalu jsou povoleny

Parameters

range	List s intervaly

5.3.3.3 void SharpIndex.Chart.TestInChart (string *sortAs*)

testuje zda zadaný řetězec obsahuje pouze povolené znaky

Parameters

sortAs	vstupní řetězec

Exceptions

ItemException	pokud vstupní řetězec obsahuje znak který není povolený

5.3.3.4 string SharpIndex.Chart.Write ( )

implementace rozhraní IWritable, vytvoří z dat v Chart řetězec pro výpis

Returns

řetězec reprezentující data v Chart

Implements SharpIndex.IWritable.

The documentation for this class was generated from the following file:

- C:/doxygen/Chart.cs

5.4 SharpIndex.Config Class Reference

načte konfigurační xml soubor, vrátí jednotlivé moduly, vyhodí vyjímku v případě chyby

Public Member Functions

- Config (String xmlPath)
  načte xml soubor dle zadané cesty a inicializuje jeho kořen
- Chart GetChart ()
  vytvoří objekt chart na základě načtených údajů z konfiguračního souboru
- Tex GetTex ()
  vytvoří objekt tex na základě načtených údajů z konfiguračního souboru
- LetterSize GetLetterSize ()
  vytvoří objekt letterSize na základě načtených údajů z konfiguračního souboru
- Order GetOrder ()
  vytvoří objekt order na základě načtených údajů z konfiguračního souboru
- Transliteration GetTransliteration ()
  vytvoří objekt transliteration na základě načtených údajů z konfiguračního souboru

5.4.1 Detailed Description

načte konfigurační xml soubor, vrátí jednotlivé moduly, vyhodí vyjímku v případě chyby

5.4.2 Constructor & Destructor Documentation

5.4.2.1 SharpIndex.Config.Config (String *xmlPath*)

načte xml soubor dle zadané cesty a inicializuje jeho kořen

Parameters

xmlPath	cesta k xml souboru

5.4.3 Member Function Documentation

5.4.3.1 Chart SharpIndex.Config.GetChart ( )

vytvoří objekt chart na základě načtených údajů z konfiguračního souboru

Returns

objekt chart

5.4.3.2 LetterSize SharpIndex.Config.GetLetterSize ( )

vytvoří objekt letterSize na základě načtených údajů z konfiguračního souboru

Returns

objekt letterSize

5.4.3.3 Order SharpIndex.Config.GetOrder ( )

vytvoří objekt order na základě načtených údajů z konfiguračního souboru

Returns

objekt order

5.4.3.4 Tex SharpIndex.Config.GetTex ( )

vytvoří objekt tex na základě načtených údajů z konfiguračního souboru

Returns

objekt tex

5.4.3.5 Transliteration SharpIndex.Config.GetTransliteration ( )

vytvoří objekt transliteration na základě načtených údajů z konfiguračního souboru

Returns

objekt translitaration

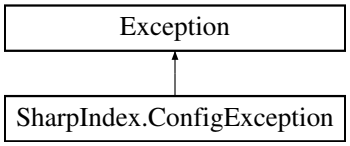
The documentation for this class was generated from the following file:

- C:/doxygen/Config.cs

5.5 SharpIndex.ConfigException Class Reference

vlastní vyjíka která může nastat při načítání konfiguračního xml souboru

Inheritance diagram for SharpIndex.ConfigException:



Public Member Functions

- ConfigException ()

  bezparametrický konstruktor vyjímky

- ConfigException (string message)

  konstruktor vyjímky se zprávou
- ConfigException (string message, Exception inner)

  konstruktor vyjímky se zprávou a předcházející vyjímkou

5.5.1 Detailed Description

vlastní vyjíka která může nastat při načítání konfiguračního xml souboru

5.5.2 Constructor & Destructor Documentation

5.5.2.1 SharpIndex.ConfigException.ConfigException ( )

bezparametrický konstruktor vyjímky

5.5.2.2 SharpIndex.ConfigException.ConfigException (string *message*)

konstruktor vyjímky se zprávou

Parameters

message	zpráva o vyjímce

5.5.2.3 SharpIndex.ConfigException.ConfigException (string *message*, Exception *inner*)

konstruktor vyjímky se zprávou a předcházející vyjímkou

Parameters

message	zpráva o vyjímce
inner	předcházející vyjímka

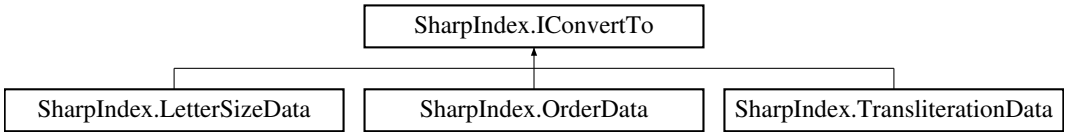
The documentation for this class was generated from the following file:

- C:/doxygen/ConfigException.cs

5.6 SharpIndex.IConvertTo Interface Reference

rozhraní které se používá u IDataStructure k definování pole znaků na které se má převádět

Inheritance diagram for SharpIndex.IConvertTo:



Public Member Functions

- char[ ] GetConvertTo ()

  vrátí pole znaků na které se má převádět

5.6.1 Detailed Description

rozhraní které se používá u IDataStructure k definování pole znaků na které se má převádět

5.6.2 Member Function Documentation

5.6.2.1 char [ ] SharpIndex.IConvertTo.GetConvertTo ( )

vrátí pole znaků na které se má převádět

Returns

pole znaků na které se má převádět

Implemented in SharpIndex.OrderData, SharpIndex.LetterSizeData, and SharpIndex.TransliterationData.

The documentation for this interface was generated from the following file:

- C:/doxygen/IConvertTo.cs

5.7 SharpIndex.IDataStructure< T > Interface Template Reference

generické rozhraní pro datové objekty, které pouzívají jako klíč 1 nebo 2 znaky

Public Member Functions

- void AddOne (char character, char[ ] charArray, String description)
  přidá do datového objektu data, kde klíčem je jeden znak
- T GetOne (char character)
  vrátí z datového objektu data pomocí zadaného klíče
- void AddTwo (char firstCharacter, char secondCharacter, char[ ] charArray, String description)
  přidá do datového objektu data, kde klíčem jsou 2 znaky
- T GetTwo (char firstCharacter, char secondCharacter)
  vrátí z datového objektu data pomocí zadaných dvou klíčů

5.7.1 Detailed Description

generické rozhraní pro datové objekty, které pouzívají jako klíč 1 nebo 2 znaky

Template Parameters

T	datový objekt

5.7.2 Member Function Documentation

5.7.2.1 void SharpIndex.IDataStructure< T >.AddOne (char *character,* char[ ] *charArray,* String *description*)

přidá do datového objektu data, kde klíčem je jeden znak

Parameters

character	klíč k datům
charArray	převodní řetězec
description	popis vložených dat

5.7.2.2 void SharpIndex.IDataStructure< T >.AddTwo (char *firstCharacter*, char *secondCharacter*, char[ ] *charArray*, String *description*)

přidá do datového objektu data, kde klíčem jsou 2 znaky

Parameters

firstCharacter	první znak
secondCharacter	druhý znak
charArray	převodní řetězec
description	popis vložených dat

5.7.2.3 T SharpIndex.IDataStructure< T >.GetOne (char *character*)

vrátí z datového objektu data pomocí zadaného klíče

Parameters

character	klíč k datům

Returns

jeden záznam dat typu T

5.7.2.4 T SharpIndex.IDataStructure< T >.GetTwo (char *firstCharacter*, char *secondCharacter*)

vrátí z datového objektu data pomocí zadaných dvou klíčů

Parameters

firstCharacter	první znak
secondCharacter	druhý znak

Returns

jeden záznam dat typu T

The documentation for this interface was generated from the following file:

- C:/doxygen/IDataStructure.cs

5.8 SharpIndex.InputFile Class Reference

načítá vstupní .idx soubor a přeposílá načtená data objektu ItemParser, v případě chyby vyhodí vyjímku InputFileException

Public Member Functions

- InputFile (string path, ItemParser itemParser)
  nastaví cestu k souboru, načte z něj data a předá je objektu itemParser, v případě chyby vyhodí vyjímku

5.8.1 Detailed Description

načítá vstupní .idx soubor a přeposílá načtená data objektu ItemParser, v případě chyby vyhodí vyjímku InputFile↩ Exception

5.8.2 Constructor & Destructor Documentation

5.8.2.1 SharpIndex.InputFile.InputFile (string *path,* ItemParser *itemParser*)

nastaví cestu k souboru, načte z něj data a předá je objektu itemParser, v případě chyby vyhodí vyjímku

Parameters

path	cesta k vstupnímu .idx souboru
itemParser	objekt kterému se předají načtená data

Exceptions

InputFileException	pokud hloubka hesla překročí 3, pokud počet parametrů v indexentry je jiný než 2, pokud řádek nezačíná příkazem indexentry

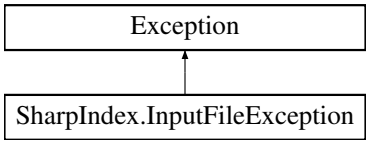
The documentation for this class was generated from the following file:

- C:/doxygen/InputFile.cs

5.9 SharpIndex.InputFileException Class Reference

vlastní vyjíka která může nastat při načítání vstupního .idx souboru

Inheritance diagram for SharpIndex.InputFileException:



Public Member Functions

- InputFileException ()
  bezparametrický konstruktor vyjímky
- InputFileException (string message)
  konstruktor vyjímky se zprávou
- InputFileException (string message, Exception inner)
  konstruktor vyjímky se zprávou a předcházející vyjímkou

5.9.1 Detailed Description

vlastní vyjíka která může nastat při načítání vstupního .idx souboru

5.9.2 Constructor & Destructor Documentation

5.9.2.1 SharpIndex.InputFileException.InputFileException ( )

bezparametrický konstruktor vyjímky

5.9.2.2 SharpIndex.InputFileException.InputFileException (string *message*)

konstruktor vyjímky se zprávou

Parameters

message	zpráva o vyjímce

5.9.2.3 SharpIndex.InputFileException.InputFileException (string *message*, Exception *inner*)

konstruktor vyjímky se zprávou a předcházející vyjímkou

Parameters

message	zpráva o vyjímce
inner	předcházející vyjímka

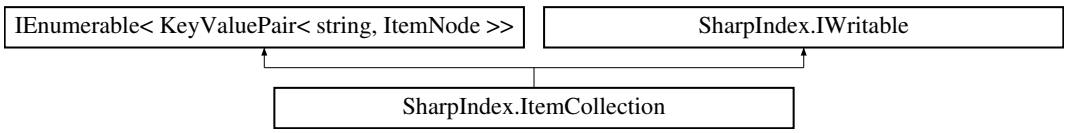
The documentation for this class was generated from the following file:

- C:/doxygen/InputFileException.cs

5.10 SharpIndex.ItemCollection Class Reference

třída reprezentující kolekci objektů ItemNode, které jsou tříděné pomoci WordComparer, proto ať lze procházet touto kolekcí pomocí foreach, implementuje generické rozhraní IEnumerable

Inheritance diagram for SharpIndex.ItemCollection:



Public Member Functions

- ItemCollection (IDataStructure< OrderData > order)

  vytvoří novou instanci ItemCollection, pomocí order inicializuje WordComparer a kolekci pro ItemNode
- void AddSortingRules (string key, char[ ] primary, char[ ] secondary)

  přidá primární a sekundární třídící pravidlo k danému klíči
- void AddItemNode (string key, ItemNode itemNode)

  přidá nový itemNode s daným klíčem
- bool ContainsNode (string key)

  vrací zda kolekce itemNodes obsahuje uzel s daným klíčem
- ItemNode GetItemNode (string key)

  vrátí z kolekce itemNode podle zadaného klíče
- IEnumerator< KeyValuePair< string, ItemNode > > GetEnumerator ()

  vrací generický IEnumerator potřebný k implementaci IEnumerable

- string Write ()

  implementace rozhraní IWritable, vrátí textovou reprezentaci všech dat obsažených v kolekci

Public Attributes

- WordComparer wordComparer

  implementace generického rozhraní IComparer použitá k třídění itemNodes

5.10.1 Detailed Description

třída reprezentující kolekci objektů ItemNode, které jsou tříděné pomoci WordComparer, proto ať lze procházet touto kolekcí pomocí foreach, implementuje generické rozhraní IEnumerable

5.10.2 Constructor & Destructor Documentation

5.10.2.1 SharpIndex.ItemCollection.ItemCollection (IDataStructure< OrderData > *order*)

vytvoří novou instanci ItemCollection, pomocí order inicializuje WordComparer a kolekci pro ItemNode

Parameters

order	objekt s posloupností grafémů

5.10.3 Member Function Documentation

5.10.3.1 void SharpIndex.ItemCollection.AddItemNode (string *key*, ItemNode *itemNode*)

přidá nový itemNode s daným klíčem

Parameters

key	klíčové slovo
itemNode	itemNode který se má přidat

5.10.3.2 void SharpIndex.ItemCollection.AddSortingRules (string *key*, char[ ] *primary*, char[ ] *secondary*)

přidá primární a sekundární třídící pravidlo k danému klíči

Parameters

key	klíčové slovo
primary	primární pravidlo
secondary	sekundární pravidlo

5.10.3.3 bool SharpIndex.ItemCollection.ContainsNode (string *key*)

vrací zda kolekce itemNodes obsahuje uzel s daným klíčem

Parameters

key	klíčové slovo

Returns

zda kolekce obsahuje uzel s daným klíčovým slovem

5.10.3.4 IEnumerator<KeyValuePair<string, ItemNode> > SharpIndex.ItemCollection.GetEnumerator ( )

vrací generický IEnumerator potřebný k implementaci IEnumerable

Returns

IEnumerator potřebný k implementaci IEnumerable

5.10.3.5 ItemNode SharpIndex.ItemCollection.GetItemNode (string *key*)

vrátí z kolekce itemNode podle zadaného klíče

Parameters

key	klíčové slovo

Returns

itemNode odpovídající zadenému klíči

5.10.3.6 string SharpIndex.ItemCollection.Write ( )

implementace rozhraní IWritable, vrátí textovou reprezentaci všech dat obsažených v kolekci

Returns

textová reprezentace všech dat obsažených v kolekci

Implements SharpIndex.IWritable.

5.10.4 Member Data Documentation

5.10.4.1 WordComparer SharpIndex.ItemCollection.wordComparer

implementace generického rozhraní IComparer použitá k třídění itemNodes

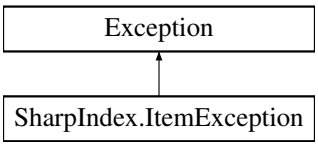
The documentation for this class was generated from the following file:

- C:/doxygen/ItemCollection.cs

5.11 SharpIndex.ItemException Class Reference

vlastní vyjíka která může nastat při načítání dat do ItemCollection nebo při třídění ItemCollection

Inheritance diagram for SharpIndex.ItemException:



Public Member Functions

- ItemException ()

  bezparametrický konstruktor vyjímky
- ItemException (string message)

  konstruktor vyjímky se zprávou
- ItemException (string message, Exception inner)

  konstruktor vyjímky se zprávou a předcházející vyjímkou

5.11.1 Detailed Description

vlastní vyjíka která může nastat při načítání dat do ItemCollection nebo při třídění ItemCollection

5.11.2 Constructor & Destructor Documentation

5.11.2.1 SharpIndex.ItemException.ItemException ( )

bezparametrický konstruktor vyjímky

5.11.2.2 SharpIndex.ItemException.ItemException (string *message*)

konstruktor vyjímky se zprávou

Parameters

message	zpráva o vyjímce

5.11.2.3 SharpIndex.ItemException.ItemException (string *message*, Exception *inner*)

konstruktor vyjímky se zprávou a předcházející vyjímkou

Parameters

message	zpráva o vyjímce
inner	předcházející vyjímka

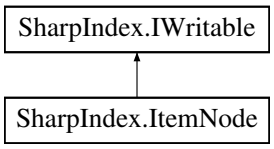
The documentation for this class was generated from the following file:

- C:/doxygen/ItemException.cs

5.12 SharpIndex.ItemNode Class Reference

třída reprezentující položku v rejstříku včetně podpoložek

Inheritance diagram for SharpIndex.ItemNode:



Public Member Functions

- ItemNode (string displayAs, Order order)

  vytvoří nový itemNode, který se bude zobrazovat v rejstříku podle zadaného parametru a inicializuje kolekci potomků pomocí order
- void AddIndex (int indexType, int indexValue)

  přidá index daného typu
- ItemCollection GetItemCollection ()

  vrátí kolekci itemů
- string Write ()

  implementace rozhraní IWritable, vrátí textovou reprezentaci dat v itemu

Public Attributes

- const int INDEX_SINGLEPAGE = 0

  konstanta která určuje že index se vztahuje pouze na jednu stranu
- const int INDEX_HYPERPAGE_LEFT = 1

  konstanta která určuje že index je levá mez intervalu stránek na které se heslo vyskytuje
- const int INDEX_HYPERPAGE_RIGHT = 2

  konstanta která určuje že index je pravá mez intervalu stránek na které se heslo vyskytuje

Properties

- string DisplayAs `[get]`

  vrací řetězec reprezentující jak se má item zobrazit v rejstříku
- List< int > SinglePageIndex `[get]`

  vrací generický List obsahující čísla stránek na kterém se item vyskytuje
- List< int[ ]> HyperPageIndex `[get]`

  vrací generický List obsahující intervaly stránek na kterém se item vyskytuje

5.12.1 Detailed Description

třída reprezentující položku v rejstříku včetně podpoložek

5.12.2 Constructor & Destructor Documentation

5.12.2.1 SharpIndex.ItemNode.ItemNode (string *displayAs*, Order *order*)

vytvoří nový itemNode, který se bude zobrazovat v rejstříku podle zadaného parametru a inicializuje kolekci potomků pomocí order

Parameters

displayAs	jak se má položka zobrazit v rejstříku
order	objekt reprezentující pořadí grafémů

5.12.3 Member Function Documentation

5.12.3.1 void SharpIndex.ItemNode.AddIndex (int *indexType*, int *indexValue*)

přidá index daného typu

Parameters

indexType	typ indexu, jedna z konstant ItemNode.INDEX_SINGLEPAGE, ItemNode.INDEX_HYPERPAGE_LEFT nebo ItemNode.INDEX_HYPERPAGE_RIGHT
indexValue	číslo stránky

Exceptions

ItemException	pokud hyperpage index není uzavřený

5.12.3.2 ItemCollection SharpIndex.ItemNode.GetItemCollection ( )

vrátí kolekci itemů

Returns

kolekce itemů

5.12.3.3 string SharpIndex.ItemNode.Write ( )

implementace rozhraní IWritable, vrátí textovou reprezentaci dat v itemu

Returns

textová reprezentace dat v itemu

Implements SharpIndex.IWritable.

5.12.4 Member Data Documentation

5.12.4.1 const int SharpIndex.ItemNode.INDEX_HYPERPAGE_LEFT = 1

konstanta která určuje že index je levá mez intervalu stránek na které se heslo vyskytuje

5.12.4.2 const int SharpIndex.ItemNode.INDEX_HYPERPAGE_RIGHT = 2

konstanta která určuje že index je pravá mez intervalu stránek na které se heslo vyskytuje

5.12.4.3 const int SharpIndex.ItemNode.INDEX_SINGLEPAGE = 0

konstanta která určuje že index se vztahuje pouze na jednu stranu

5.12.5 Property Documentation

5.12.5.1 string SharpIndex.ItemNode.DisplayAs [get]

vrací řetězec reprezentující jak se má item zobrazit v rejstříku

5.12.5.2 List<int[]> SharpIndex.ItemNode.HyperPageIndex [get]

vrací generický List obsahující intervaly stránek na kterém se item vyskytuje

5.12.5.3 List<int> SharpIndex.ItemNode.SinglePageIndex [get]

vrací generický List obsahující čísla stránek na kterém se item vyskytuje

The documentation for this class was generated from the following file:

- C:/doxygen/ItemNode.cs

5.13 SharpIndex.ItemParser Class Reference

třída se stará o nastavení třídících pravidel a správné vložení dat do ItemCollection

Public Member Functions

- ItemParser (ItemCollection itemCollection, Chart chart, Tex tex, LetterSize letterSize, Order order, Transliteration transliteration)

  konstroktor který inicializuje potřebné objekty
- void AddEntryDepth1 (string sortAs1, string displayAs1, int indexType, int index)

  přidá heslo hloubky 1 do kolekce
- void AddEntryDepth2 (string sortAs1, string displayAs1, string sortAs2, string displayAs2, int indexType, int index)

  přidá heslo hloubky 2 do kolekce
- void AddEntryDepth3 (string sortAs1, string displayAs1, string sortAs2, string displayAs2, string sortAs3, string displayAs3, int indexType, int index)

  přidá heslo hloubky 3 do kolekce
- char[] GetKey (string sortAs)

  vrátí klíč z hesla podle kterého se má řadit
- char[] ConvertWord< T, U > (ref T dataObject, char[] word)

  generická metoda která pomocí objektu který implementuje rozhraní IDataStructure převede dané slovo jiné (vše na malá písmena, transliterace, primární třídící pravidlo), pro to ať je jasné na jaký řetězec datová struktura slovo převádí, musí data implementovat rozhraní IConvertTo

Public Attributes

- Chart chart

  objekt ke kontrole povolených znaků ve slově
- Tex tex

  objekt k převodu TeXových příkazů
- LetterSize letterSize

  objekt k převodu na malá písmena
- Order order

objekt definující pořadí grafémů a primární třídící pravidlo

- Transliteration transliteration

  objekt k provádějící translitaraci

- ItemCollection itemCollection

  kolekce sloužící k reprezentaci položek v rejstříku

5.13.1 Detailed Description

třída se stará o nastavení třídících pravidel a správné vložení dat do ItemCollection

5.13.2 Constructor & Destructor Documentation

5.13.2.1 SharpIndex.ItemParser.ItemParser (ItemCollection *itemCollection,* Chart *chart,* Tex *tex,* LetterSize *letterSize,* Order *order,* Transliteration *transliteration*)

konstroktor který inicializuje potřebné objekty

Parameters

itemCollection	kolekce sloužící k reprezentaci položek v rejstříku
chart	objekt ke kontrole povolených znaků ve slově
tex	objekt k převodu TeXových příkazů
letterSize	objekt k převodu na malá písmena
order	objekt definující pořadí grafémů a primární třídící pravidlo
transliteration	objekt potřebný k translitaraci

5.13.3 Member Function Documentation

5.13.3.1 void SharpIndex.ItemParser.AddEntryDepth1 (string *sortAs1,* string *displayAs1,* int *indexType,* int *index*)

přidá heslo hloubky 1 do kolekce

Parameters

sortAs1	jak se má heslo řadit
displayAs1	jak se má heslo zobrazit
indexType	typ indexu, jedna z konstant ItemNode.INDEX_SINGLEPAGE, ItemNode.INDEX_HYPERPAGE_LEFT nebo ItemNode.INDEX_HYPERPAGE_RIGHT
index	číslo stránky

5.13.3.2 void SharpIndex.ItemParser.AddEntryDepth2 (string *sortAs1,* string *displayAs1,* string *sortAs2,* string *displayAs2,* int *indexType,* int *index*)

přidá heslo hloubky 2 do kolekce

Parameters

sortAs1	jak se má heslo v hloubce 1 řadit
displayAs1	jak se má heslo v hloubce 1 zobrazit
sortAs2	jak se má podheslo v hloubce 2 řadit
displayAs2	jak se má podheslo v hloubce 2 zobrazit

indexType	typ indexu, jedna z konstant ItemNode.INDEX_SINGLEPAGE, ItemNode.INDEX_HYPERPAGE_LEFT nebo ItemNode.INDEX_HYPERPAGE_RIGHT
index	číslo stránky

5.13.3.3 void SharpIndex.ItemParser.AddEntryDepth3 (string *sortAs1,* string *displayAs1,* string *sortAs2,* string *displayAs2,* string *sortAs3,* string *displayAs3,* int *indexType,* int *index*)

přidá heslo hloubky 3 do kolekce

Parameters

sortAs1	jak se má heslo v hloubce 1 řadit
displayAs1	jak se má heslo v hloubce 1 zobrazit
sortAs2	jak se má podheslo v hloubce 2 řadit
displayAs2	jak se má podheslo v hloubce 2 zobrazit
sortAs3	jak se má podpodheslo v hloubce 3 řadit
displayAs3	jak se má podpodheslo v hloubce 3 zobrazit
indexType	typ indexu, jedna z konstant ItemNode.INDEX_SINGLEPAGE, ItemNode.INDEX_HYPERPAGE_LEFT nebo ItemNode.INDEX_HYPERPAGE_RIGHT
index	číslo stránky

5.13.3.4 char [ ] SharpIndex.ItemParser.ConvertWord< T, U > (ref T *dataObject,* char[ ] *word*)

generická metoda která pomocí objektu který implementuje rozhraní IDataStructure převede dané slovo jiné (vše na malá písmena, transliterace, primární třídící pravidlo), pro to ať je jasné na jaký řetězec datová struktura slovo převádí, musí data implementovat rozhraní IConvertTo

Template Parameters

T	objekt ve kterém jsou uložená data pro převod
U	jeden záznam dat objektu T

Parameters

dataObject	objekt ve kterém jsou uložená data pro převod
word	slovo které se má převést

Returns

převedné slovo

Type Constraints

***T* : *IDataStructure*<*U*>**

U* : *IConvertTo

5.13.3.5 char [ ] SharpIndex.ItemParser.GetKey (string *sortAs*)

vrátí klíč z hesla podle kterého se má řadit

Parameters

sortAs	jak se má dané heslo řadit

Returns

klíč k heslu

5.13.4 Member Data Documentation

5.13.4.1 Chart SharpIndex.ItemParser.chart

objekt ke kontrole povolených znaků ve slově

5.13.4.2 ItemCollection SharpIndex.ItemParser.itemCollection

kolekce sloužící k reprezentaci položek v rejstříku

5.13.4.3 LetterSize SharpIndex.ItemParser.letterSize

objekt k převodu na malá písmena

5.13.4.4 Order SharpIndex.ItemParser.order

objekt definující pořadí grafémů a primární třídící pravidlo

5.13.4.5 Tex SharpIndex.ItemParser.tex

objekt k převodu TeXových příkazů

5.13.4.6 Transliteration SharpIndex.ItemParser.transliteration

objekt k provádějící translitaraci

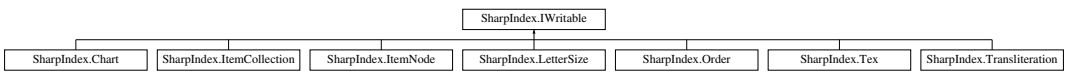
The documentation for this class was generated from the following file:

- C:/doxygen/ItemParser.cs

5.14 SharpIndex.IWritable Interface Reference

rozhraní které používá třída Log k výpisu dat

Inheritance diagram for SharpIndex.IWritable:



Public Member Functions

- string Write ()

  vypíše řetězec reprezentující data v objektu

5.14.1 Detailed Description

rozhraní které používá třída Log k výpisu dat

5.14.2 Member Function Documentation

5.14.2.1 string SharpIndex.IWritable.Write ( )

vypíše řetězec reprezentující data v objektu

Returns

Implemented in SharpIndex.Tex, SharpIndex.ItemNode, SharpIndex.ItemCollection, SharpIndex.Order, SharpIndex.Transliteration, SharpIndex.LetterSize, and SharpIndex.Chart.

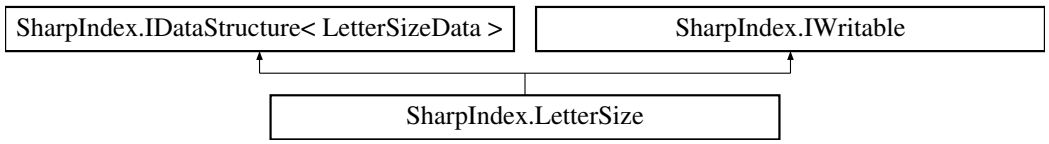
The documentation for this interface was generated from the following file:

- C:/doxygen/IWritable.cs

5.15 SharpIndex.LetterSize Class Reference

třída která uchovává data pro převod velkých písmen na malá

Inheritance diagram for SharpIndex.LetterSize:



Public Member Functions

- LetterSize ()

  inicializuje objekt LetterSize a jeho privátní proměnné
- void AddOne (char upper, char[ ] lower, string desc)

  implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené jedním znakem
- LetterSizeData GetOne (char letter)

  implementace rozhraní IDataStructure, vrátí data o písmenu tvořeném jedním znakem, pokud daný znak není ve slovníku, vrátí null
- void AddTwo (char firstUpper, char secondUpper, char[ ] lower, string desc)

  implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené dvěma znaky
- LetterSizeData GetTwo (char firstLetter, char secondLetter)

  implementace rozhraní IDataStructure, vrátí data o písmenu tvořeném dvěma znaky, pokud daný znak není ve slovníku, vrátí null
- string Write ()

  implementace rozhraní IWritable, vytvoří z dat v LetterSize řetezec pro výpis

5.15.1 Detailed Description

třída která uchovává data pro převod velkých písmen na malá

5.15.2 Constructor & Destructor Documentation

5.15.2.1 SharpIndex.LetterSize.LetterSize ( )

inicializuje objekt LetterSize a jeho privátní proměnné

5.15.3 Member Function Documentation

5.15.3.1 void SharpIndex.LetterSize.AddOne (char *upper*, char[ ] *lower*, string *desc*)

implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené jedním znakem

Parameters

upper	velké písmeno
lower	malé písmeno
desc	popis písmene

5.15.3.2 void SharpIndex.LetterSize.AddTwo (char *firstUpper*, char *secondUpper*, char[ ] *lower*, string *desc*)

implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené dvěma znaky

Parameters

firstUpper	první znak velkého písmene
secondUpper	druhý znak velkého písmene
lower	malé písmeno
desc	popis písmene

5.15.3.3 LetterSizeData SharpIndex.LetterSize.GetOne (char *letter*)

implementace rozhraní IDataStructure, vrátí data o písmenu tvořeném jedním znakem, pokud daný znak není ve slovníku, vrátí null

Parameters

letter	znak který se má převést

Returns

data odpovídající malému písmenu nebo null

5.15.3.4 LetterSizeData SharpIndex.LetterSize.GetTwo (char *firstLetter*, char *secondLetter*)

implementace rozhraní IDataStructure, vrátí data o písmenu tvořeném dvěma znaky, pokud daný znak není ve slovníku, vrátí null

Parameters

firstLetter	první znak písmene ktere se má převést
secondLetter	druhy znak písmene ktere se má převést

Returns

data odpovídající malému písmenu nebo null

5.15.3.5 string SharpIndex.LetterSize.Write ( )

implementace rozhraní IWritable, vytvoří z dat v LetterSize řetezec pro výpis

Returns

řetezec reprezentující data v LetterSize

Implements SharpIndex.IWritable.

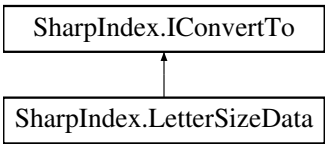
The documentation for this class was generated from the following file:

- C:/doxygen/LetterSize.cs

5.16 SharpIndex.LetterSizeData Class Reference

data o převodu velkého písmene na malé

Inheritance diagram for SharpIndex.LetterSizeData:



Public Member Functions

- LetterSizeData (char[ ] letter, string desc)
  vytvoří záznam dat o převodu písmene na malé
- char[ ] GetConvertTo ()
  implementace rozhraní IConvertTo, určuje že pro převod se má použít proměnná letter

Properties

- char[ ] Letter `[get]`
  vrací malé písmeno
- string Description `[get]`
  vrací popis o jaké písmeno se jedná

5.16.1 Detailed Description

data o převodu velkého písmene na malé

5.16.2 Constructor & Destructor Documentation

5.16.2.1 SharpIndex.LetterSizeData.LetterSizeData (char[ ] *letter,* string *desc*)

vytvoří záznam dat o převodu písmene na malé

Parameters

letter	malé písmeno
desc	popis o jaké písmeno se jedná

5.16.3 Member Function Documentation

5.16.3.1 char [ ] SharpIndex.LetterSizeData.GetConvertTo ( )

implementace rozhraní IConvertTo, určuje že pro převod se má použít proměnná letter

Returns

malé písmeno

Implements SharpIndex.IConvertTo.

5.16.4 Property Documentation

5.16.4.1 string SharpIndex.LetterSizeData.Description [get]

vrací popis o jaké písmeno se jedná

5.16.4.2 char [ ] SharpIndex.LetterSizeData.Letter [get]

vrací malé písmeno

The documentation for this class was generated from the following file:

- C:/doxygen/LetterSizeData.cs

5.17 SharpIndex.Log Class Reference

třída shromažďuje data která se mají zapsat do logu

Public Member Functions

- Log ()

  bezparametrický konstruktor který inicializuje proměnné
- void AddWritableObject (IWritable writableObject)

  přidá objekt který implementuje rozhraní IWritable do logu
- void WriteIntoConsole (string line)

  metoda zapisuje text do konzole a ukládá kopii zapsaného textu do třídní proměnné
- StringBuilder GetLog ()

  projde všechny objekty přidané do logu a přidá jejich textovou reprezentaci do objektu StringBuilder, nakonec přidá i záznam z konzole a vrátí tento objekt StringBuilder

Properties

- bool WriteLog [get, set]

  vrací a nastavuje zda se má zapsat log do souboru

5.17.1 Detailed Description

třída shromazďuje data která se mají zapsat do logu

5.17.2 Constructor & Destructor Documentation

5.17.2.1 SharpIndex.Log.Log ( )

bezparametrický konstruktor který inicializuje proměnné

5.17.3 Member Function Documentation

5.17.3.1 void SharpIndex.Log.AddWritableObject (IWritable *writableObject*)

přidá objekt který implementuje rozhraní IWritable do logu

Parameters

writableObject	

5.17.3.2 StringBuilder SharpIndex.Log.GetLog ( )

projde všechny objekty přidané do logu a přidá jejich textovou reprezentaci do objektu StringBuilder, nakonec přidá i záznam z konzole a vrátí tento objekt StringBuilder

Returns

StringBuilder obsahující všechny textové reprezentace přidaných objektů včetně konzole

5.17.3.3 void SharpIndex.Log.WriteIntoConsole (string *line*)

metoda zapisuje text do konzole a ukládá kopii zapsaného textu do třídní proměnné

Parameters

line	

5.17.4 Property Documentation

5.17.4.1 bool SharpIndex.Log.WriteLog [get],[set]

vrací a nastavuje zda se má zapsat log do souboru

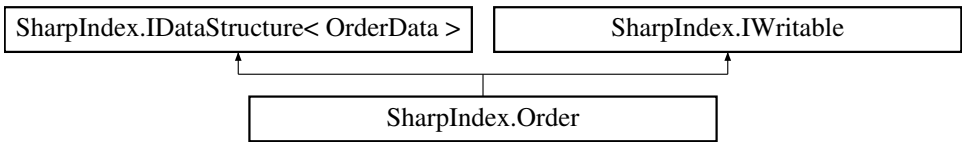
The documentation for this class was generated from the following file:

- C:/doxygen/Log.cs

5.18 SharpIndex.Order Class Reference

třída shromažďující data o pořadí grafémů a primárním třídícím pravidle

Inheritance diagram for SharpIndex.Order:



Public Member Functions

- Order ()

  inicializuje slovníky a nastaví počáteční prioritu
- void AddOne (char grapheme, char[ ] primary, String description)

  implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené jedním znakem
- OrderData GetOne (char character)

  implementace rozhraní IDataStructure, vrátí data o grafému tvořeném jedním znakem, pokud znak není nalezen, vyhodí vyjímku
- void AddTwo (char firstCharacter, char secondCharacter, char[ ] primary, string desc)

  implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené dvěma znaky
- OrderData GetTwo (char firstCharacter, char secondCharacter)

  implementace rozhraní IDataStructure, vrátí data o grafému tvořeném dvěma znaky
- string Write ()

  implementace rozhraní IWritable, vytvoří z dat v Conversions řetězec pro výpis

5.18.1 Detailed Description

třída shromažďující data o pořadí grafémů a primárním třídícím pravidle

5.18.2 Constructor & Destructor Documentation

5.18.2.1 SharpIndex.Order.Order ( )

inicializuje slovníky a nastaví počáteční prioritu

5.18.3 Member Function Documentation

5.18.3.1 void SharpIndex.Order.AddOne (char *grapheme*, char[ ] *primary*, String *description*)

implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené jedním znakem

Parameters

grapheme	grafém z abecedy
primary	jak se má grafém primárně řadit
description	popis přidaného grafému

5.18.3.2 void SharpIndex.Order.AddTwo (char *firstCharacter*, char *secondCharacter*, char[ ] *primary*, string *desc*)

implementace rozhraní IDataStructure, přidá do slovníku písmeno tvořené dvěma znaky

Parameters

firstCharacter	první znak grafému z abecedy
secondCharacter	druhý znak grafému z abecedy
primary	jak se má grafém primárně řadit
desc	popis přidaného grafému

5.18.3.3 OrderData SharpIndex.Order.GetOne (char *character*)

implementace rozhraní IDataStructure, vrátí data o grafému tvořeném jedním znakem, pokud znak není nalezen, vyhodí vyjímku

Parameters

character	hledaný grafém

Returns

data odpovídající danému grafému

Exceptions

ConfigException	pokud hledaný grafém není nalezen

5.18.3.4 OrderData SharpIndex.Order.GetTwo (char *firstCharacter,* char *secondCharacter*)

implementace rozhraní IDataStructure, vrátí data o grafému tvořeném dvěma znaky

Parameters

firstCharacter	první znak grafému
secondCharacter	druhý znak grafému

Returns

data odpovídající danému grafému

5.18.3.5 string SharpIndex.Order.Write ( )

implementace rozhraní IWritable, vytvoří z dat v Conversions řetězec pro výpis

Returns

řetězec reprezentující data v Order

Implements SharpIndex.IWritable.

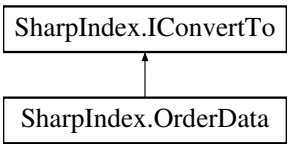
The documentation for this class was generated from the following file:

- C:/doxygen/Order.cs

5.19 SharpIndex.OrderData Class Reference

data o pořadí a třídění grafému

Inheritance diagram for SharpIndex.OrderData:



Public Member Functions

- OrderData (char[ ] primary, int priority, string description)

  vytvoří data o grafému a jeho pořadí v abecedě
- char[ ] GetConvertTo ()

  implementace rozhraní IConvertTo, určuje že pro převod se má použí proměnná primary

Properties

- char[ ] Primary `[get]`

  vrací jak se má grafém primárně řadit
- int Priority `[get]`

  vrací prioritu grafému, menší číslo je dřív v abecedě
- string Description `[get]`

  vrací popis daného grafému

5.19.1 Detailed Description

data o pořadí a třídění grafému

5.19.2 Constructor & Destructor Documentation

5.19.2.1 SharpIndex.OrderData.OrderData (char[ ] *primary*, int *priority*, string *description*)

vytvoří data o grafému a jeho pořadí v abecedě

Parameters

primary	jak se má grafém primárně řadit
priority	priorita grafému, menší číslo je dřív v abecedě
description	popis daného grafému

5.19.3 Member Function Documentation

5.19.3.1 char [ ] SharpIndex.OrderData.GetConvertTo ( )

implementace rozhraní IConvertTo, určuje že pro převod se má použí proměnná primary

Returns

Implements SharpIndex.IConvertTo.

5.19.4 Property Documentation

5.19.4.1 string SharpIndex.OrderData.Description [get]

vrací popis daného grafému

5.19.4.2 char [ ] SharpIndex.OrderData.Primary [get]

vrací jak se má grafém primárně řadit

5.19.4.3 int SharpIndex.OrderData.Priority [get]

vrací prioritu grafému, menší číslo je dřív v abecedě

The documentation for this class was generated from the following file:

- C:/doxygen/OrderData.cs

5.20 SharpIndex.OutputFile Class Reference

třída starající se o zápis dat do souboru

Public Member Functions

- OutputFile (string path, ItemCollection itemCollection)
  vytvoří .ind soubor z kolekce ItemCollection
- OutputFile (Log log)
  vytvoří sharpindex.log soubor s daty z oblektu log

Public Attributes

- const string LOG_PATH = "sharpindex.log"
  cesta k logovacímu souboru

5.20.1 Detailed Description

třída starající se o zápis dat do souboru

5.20.2 Constructor & Destructor Documentation

5.20.2.1 SharpIndex.OutputFile.OutputFile (string *path,* ItemCollection *itemCollection*)

vytvoří .ind soubor z kolekce ItemCollection

Parameters

path	cesta k výstupnímu .ind souboru

itemCollection	kolekce ze které bude .ind soubor vytvořen

5.20.2.2 SharpIndex.OutputFile.OutputFile (Log *log*)

vytvoří sharpindex.log soubor s daty z oblektu log

Parameters

log	objekt s daty které se mají zapsat do souboru

5.20.3 Member Data Documentation

5.20.3.1 const string SharpIndex.OutputFile.LOG_PATH = "sharpindex.log"

cesta k logovacímu souboru

The documentation for this class was generated from the following file:

- C:/doxygen/OutputFile.cs

5.21 SharpIndex.SharpIndex Class Reference

hlavní spouštěcí třída programu sharpindex, řídí proces vytváření rejstříku, zachytává případné vyjímky

Public Member Functions

- SharpIndex ()

  konstruktor oznámí start programu do konzole a zapíše do logu

5.21.1 Detailed Description

hlavní spouštěcí třída programu sharpindex, řídí proces vytváření rejstříku, zachytává případné vyjímky

5.21.2 Constructor & Destructor Documentation

5.21.2.1 SharpIndex.SharpIndex.SharpIndex ( )

konstruktor oznámí start programu do konzole a zapíše do logu

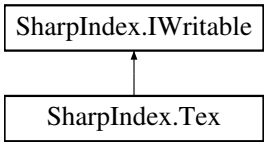
The documentation for this class was generated from the following file:

- C:/doxygen/SharpIndex.cs

5.22 SharpIndex.Tex Class Reference

třída převádí TeXové příkazy ať je možné hesla správně třídit

Inheritance diagram for SharpIndex.Tex:



Public Member Functions

- Tex ()

  bezparametrický konstruktor, inicializuje slovník pro data
- void AddTexCommand (string command, string convertTo, string desc)

  přidá příkaz do slovníku
- string ConvertTexCommands (string word)

  převede všechny známé příkazy ve formátu příkaz{parametr} na odpovídající převodní řetězce
- string Write ()

  implementace rozhraní IWritable, vrátí textovou reprezentaci dat v tomto objektu

5.22.1 Detailed Description

třída převádí TeXové příkazy ať je možné hesla správně třídit

5.22.2 Constructor & Destructor Documentation

5.22.2.1 SharpIndex.Tex.Tex ( )

bezparametrický konstruktor, inicializuje slovník pro data

5.22.3 Member Function Documentation

5.22.3.1 void SharpIndex.Tex.AddTexCommand (string *command,* string *convertTo,* string *desc*)

přidá příkaz do slovníku

Parameters

command	TeXový příkaz
convertTo	na co se má příkaz převést
desc	popis příkazu

5.22.3.2 string SharpIndex.Tex.ConvertTexCommands (string *word*)

převede všechny známé příkazy ve formátu příkaz{parametr} na odpovídající převodní řetězce

Parameters

word	slovo ve kterém se mají hledat TeXové příkazy

Returns

převedené slovo

Exceptions

ItemException	pokud příkaz nemá parametr, pokud příkazu chybí pravá složená závorka, pokud příkaz není ve slovníku

5.22.3.3 string SharpIndex.Tex.Write ( )

implementace rozhraní IWritable, vrátí textovou reprezentaci dat v tomto objektu

Returns

Implements SharpIndex.IWritable.

The documentation for this class was generated from the following file:

- C:/doxygen/Tex.cs

5.23 SharpIndex.TexData Class Reference

třída obsahující data potřebná pro převod TeXových příkazů

Public Member Functions

- TexData (string convertTo, string description)
  vytvoří datový záznam o TeXovém příkazu

Properties

- string ConvertTo `[get]`
  vrací na jaký řetězec se má příkaz převést
- string Description `[get]`
  vrací popis daného příkazu

5.23.1 Detailed Description

třída obsahující data potřebná pro převod TeXových příkazů

5.23.2 Constructor & Destructor Documentation

5.23.2.1 SharpIndex.TexData.TexData (string *convertTo*, string *description*)

vytvoří datový záznam o TeXovém příkazu

Parameters

convertTo	na jaký řetězec se má příkaz převést

description	popis příkazu

5.23.3 Property Documentation

5.23.3.1 string SharpIndex.TexData.ConvertTo [get]

vrací na jaký řetězec se má příkaz převést

5.23.3.2 string SharpIndex.TexData.Description [get]

vrací popis daného příkazu

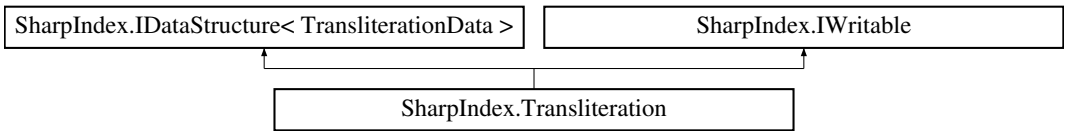
The documentation for this class was generated from the following file:

- C:/doxygen/TexData.cs

5.24 SharpIndex.Transliteration Class Reference

třída provádí transliteraci, napr. nemecke ostre "s" na "ss", uchovává data k tomu potřebná

Inheritance diagram for SharpIndex.Transliteration:



Public Member Functions

- Transliteration ()
  inicializuje objekt Transliteration a jeho privátní proměnné
- void AddOne (char character, char[ ] convertTo, String description)
  implementace rozhraní IDataStructure, přidá do slovníku transliteraci tvořenou jedním znakem
- TransliterationData GetOne (char character)
  implementace rozhraní IDataStructure, vrátí data o transliteraci, pokud není ve slovníku, vrátí null
- void AddTwo (char firstCharacter, char secondCharacter, char[ ] convertTo, String description)
  implementace rozhraní IDataStructure, přidá do slovníku transliteraci tvořenou dvěma znaky
- TransliterationData GetTwo (char firstCharacter, char secondCharacter)
  implementace rozhraní IDataStructure, vrátí data o transliteraci, pokud není ve slovníku, vrátí null
- string Write ()
  implementace rozhraní IWritable, vytvoří z dat v Transliterations řetězec pro výpis

5.24.1 Detailed Description

třída provádí transliteraci, napr. nemecke ostre "s" na "ss", uchovává data k tomu potřebná

5.24.2 Constructor & Destructor Documentation

5.24.2.1 SharpIndex.Transliteration.Transliteration ( )

inicializuje objekt Transliteration a jeho privátní proměnné

5.24.3 Member Function Documentation

5.24.3.1 void SharpIndex.Transliteration.AddOne (char *character,* char[ ] *convertTo,* String *description*)

implementace rozhraní IDataStructure, přidá do slovníku transliteraci tvořenou jedním znakem

Parameters

character	znak
convertTo	pole znaků na ktere se má převést
description	popis transliterace

5.24.3.2 void SharpIndex.Transliteration.AddTwo (char *firstCharacter,* char *secondCharacter,* char[ ] *convertTo,* String *description*)

implementace rozhraní IDataStructure, přidá do slovníku transliteraci tvořenou dvěma znaky

Parameters

firstCharacter	první znak
secondCharacter	druhý znak
convertTo	pole znaků na které se má převést
description	popis transliterace

5.24.3.3 TransliterationData SharpIndex.Transliteration.GetOne (char *character*)

implementace rozhraní IDataStructure, vrátí data o transliteraci, pokud není ve slovníku, vrátí null

Parameters

character	znak který se má převést

Returns

data odpovídající transliteraci nebo null

5.24.3.4 TransliterationData SharpIndex.Transliteration.GetTwo (char *firstCharacter,* char *secondCharacter*)

implementace rozhraní IDataStructure, vrátí data o transliteraci, pokud není ve slovníku, vrátí null

Parameters

firstCharacter	první znak převáděného znaku
secondCharacter	druhý znak převáděného znaku

Returns

data odpovídající transliteraci nebo null

5.24.3.5 string SharpIndex.Transliteration.Write ( )

implementace rozhraní IWritable, vytvoří z dat v Transliterations řetězec pro výpis

Returns

řetězec reprezentující data v Transliterations

Implements SharpIndex.IWritable.

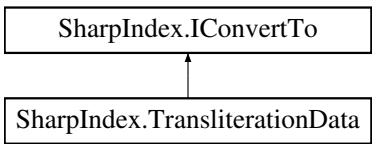
The documentation for this class was generated from the following file:

- C:/doxygen/Transliteration.cs

5.25 SharpIndex.TransliterationData Class Reference

data o transliteraci

Inheritance diagram for SharpIndex.TransliterationData:



Public Member Functions

- TransliterationData (char[ ] convertTo, string desc)
  vytvoří nová data o transliteraci
- char[ ] GetConvertTo ()
  implementace rozhraní IConvertTo, určuje že pro převod se má použít proměnná convertTo

Properties

- char[ ] ConvertTo `[get]`
  vrátí na jaký řetězec se má převádět
- string Description `[get]`
  vrátí popis transliterace

5.25.1 Detailed Description

data o transliteraci

5.25.2 Constructor & Destructor Documentation

5.25.2.1 SharpIndex.TransliterationData.TransliterationData (char[ ] *convertTo*, string *desc*)

vytvoří nová data o transliteraci

Parameters

convertTo	na jaký řetězec se má převádět

desc	popis transliterace

5.25.3 Member Function Documentation

5.25.3.1 char [ ] SharpIndex.TransliterationData.GetConvertTo ( )

implementace rozhraní IConvertTo, určuje že pro převod se má použít proměnná convertTo

Returns

Implements SharpIndex.IConvertTo.

5.25.4 Property Documentation

5.25.4.1 char [ ] SharpIndex.TransliterationData.ConvertTo [get]

vrátí na jaký řetězec se má převádět

5.25.4.2 string SharpIndex.TransliterationData.Description [get]

vrátí popis transliterace

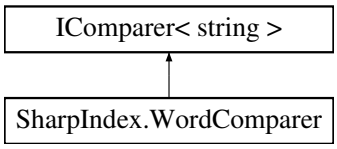
The documentation for this class was generated from the following file:

- C:/doxygen/TransliterationData.cs

5.26 SharpIndex.WordComparer Class Reference

třída se stará o třídění prvků dle nastavených primárních a sekundárních pravidel

Inheritance diagram for SharpIndex.WordComparer:



Public Member Functions

- WordComparer (IDataStructure< OrderData > order)
  vytvoří comparer s pořadím znaků jež jsou definované v order
- void AddRule (string key, char[ ] primary, char[ ] secondary)
  přidá k danému klíči jak se řadí primárně a jak sekundárně
- char[ ][ ] GetRules (string key)
  vrátí jak se daný klíč řadí primárně a jak sekundárně
- int Compare (string key1, string key2)
  implementace generického rozhraní IComparer, porovná 2 klíče

Public Attributes

- const int PRIMARY = 0

  pozice s primárním řazením
- const int SECONDARY = 1

  pozice se sekundárním řazením

5.26.1 Detailed Description

třída se stará o třídění prvků dle nastavených primárních a sekundárních pravidel

5.26.2 Constructor & Destructor Documentation

5.26.2.1 SharpIndex.WordComparer.WordComparer (IDataStructure< OrderData > *order*)

vytvoří comparer s pořadím znaků jež jsou definované v order

Parameters

order	objekt který definuje pořadí v abecedním řazení

5.26.3 Member Function Documentation

5.26.3.1 void SharpIndex.WordComparer.AddRule (string *key*, char[ ] *primary*, char[ ] *secondary*)

přidá k danému klíči jak se řadí primárně a jak sekundárně

Parameters

key	klíč
primary	primární řazení
secondary	sekundární řazení

5.26.3.2 int SharpIndex.WordComparer.Compare (string *key1*, string *key2*)

implementace generického rozhraní IComparer, porovná 2 klíče

Parameters

key1	první klíč
key2	druhý klíč

Returns

-1 pokud první je před druhým, 0 pokud jsou si rovny, jinak 1

5.26.3.3 char [ ][ ] SharpIndex.WordComparer.GetRules (string *key*)

vrátí jak se daný klíč řadí primárně a jak sekundárně

Parameters

key	klíč

Returns

primární a sekundární řazení

5.26.4 Member Data Documentation

5.26.4.1 const int SharpIndex.WordComparer.PRIMARY = 0

pozice s primárním řazením

5.26.4.2 const int SharpIndex.WordComparer.SECONDARY = 1

pozice se sekundárním řazením

The documentation for this class was generated from the following file:

- C:/doxygen/WordComparer.cs

Index